(1) i.) p is prime, so by definition, its only factors are 1 and itself. Therefore, for any $0<n<p$, $\gcd(n,p) = 1$. These are exactly the set restraints of φ, so the total size of the set is $p-1$ - what φ returns ✓

ii.) We need to count how many numbers are not coprime to pq, i.e. divisible by p or q From 1 to pq.

Numbers divisible by p: ✓

$$\underbrace{p, 2p, \ldots, (q-1)p, qp}_{q \text{ numbers}}$$

Swapping p and q gives us p numbers divisible by q. ✓

But pq counted twice, so total

number of non-coprimes are

$p + q - 1$.

Total number of ints from 1 to pq is pq.

So total number of numbers coprime to pq, $\varphi(pq)$ is

$$pq - (p + q - 1)$$

$$= pq - p - q + 1$$

$$= (p-1)(q-1)$$

Very good explanation!

iii) Total numbers from 1 to p^2

$= p^2$

Numbers divisible by p, i.e. not coprime and not returned by $\varphi(p^2)$:

$$p, 2p, 3p, \ldots, (p-1)p, p^2$$

p numbers

So $\varphi(p^2)$ returns $p^2 - p$
which is $p(p-1)$ ✓

b) From the definition of a group, each $g \in G$ should have an inverse. That means that $g \bmod pq$ must be invertible. From Euclid's algorithm, we know this can only happen if g and pq are coprime - so $\gcd(g, pq) = 1$ ✓

Therefore we need to take out all multiples of p and q. As shown in 1 a ii), the size is $(p-1)(q-1)$ ✓

(2) a) $69 \cdot 73 \pmod{1000}$

$q = 69 \qquad p = 73$

↓

1001101

$i=0, q_i = 1$:

$2^0 \times 73 = 73$

$i=1, q_i = 0$:

$2^1 \times 73 = 146$

$i=2, q_i = 1$:

$2^2 \times 73 = 292$

$i=3, q_i = 0$:

$2^3 \times 73 = 584$

$i=4, q_i=0$:

$2^4 \times 73 = 168$

$i=5, q_i=0$:

$2^5 \times 73 = 336$

$i=6, q_i=1$:

$2^6 \times 73 = 672$

Adding all values where $q_i=1$,

$73+292+672 = 37 \pmod{1000}$

b.) $2047 = 2048 - 1$,

which is useful as 2048

is 2^{11}

So compute $2^{11} \cdot 7897 - 7897$ instead, which just involves repeated doubling, but leave out adding $2^0 \times 7897$

Yes this is correct but here is a simpler explanation:

- Bitshift 7897 left 11 times (7897 << 11)
- Subtract the result by 7897

(3) a) $p = 37$, $g = 2$, $sk_A = 7$

i) $ss = 9^7 \pmod{37}$

$7 \rightarrow 0111$

$9^{2^0} = 9$

$9^{2^1} = 81 \equiv 7$

$9^{2^2} = 12$

$9 \times 7 \equiv 26$

$26 \times 12 \equiv 16$

so $ss = 16 \pmod{37}$

ii) $13 \times ss^{-1} = m$ (enc)

You should show oyour working out

Use Euclid's algo to find ss^{-1}, which is 7 mod 37

$13 \times 7 = 17 \pmod{37}$

b) $7 = ss \times 8 \pmod{37}$

enc = ss * m

so you should have done enc * m^-1 to find ss

$8 \times 1 = 8$

$8 \times 2 = 16$

$8 \times 3 = 24$

$8 \times 8 = 27$

$8 \times 9 = 35$

$8 \times 10 = 6$

$8 \times 16 = 17$

$8 \times 17 = 25$

$8 \times 18 = 33$

$8 \times 4 = 32$

$8 \times 5 = 3$

$8 \times 6 = 11$

$8 \times 7 = 19$

$8 \times 11 = 14$

$8 \times 12 = 22$

$8 \times 13 = 30$

$8 \times 14 = 1$

$8 \times 15 = 9$

$8 \times 19 = 4$

$8 \times 20 = 12$

$8 \times 21 = 20$

$8 \times 22 = 28$

$8 \times 23 = 36$

$8 \times 24 = 7$

So $ss = 24$, which can be calculated in $O(p)$

(4) a) If $e = 1$, then

$c = m^1$ so the ciphertext is just the message! No encryption whatsoever

b.) e might not necessarily be coprime to $\varphi(n)$ when set to 2. This means it might not have an inverse, so it can't produce d,

part of the private key.

(5) a.)

Good Code

```
sage: G = Zmod(31)
....: values = []
....: ex = 0
....: while G(2^ex) not in values:
....:     values.append(G(2^ex))
....:     ex += 1
....: print(values)
[1, 2, 4, 8, 16]
```

Only 5 values generated using 2, so not a generator

b.)

```
sage: G = Zmod(31)
....: ex = 0
....: gen = 2
....: found_gen = False
....: while not found_gen:
....:     values = []
....:     while G(gen^ex) not in values:
....:         values.append(G(gen^ex))
....:         ex += 1
```

```
....:         if len(values) == 30:
....:             found_gen = True
....:         else:
....:             gen += 1
....: print(f"Found generator: {gen}")
Found generator: 3
```

So 3 is a generator ✓

c.)

```
sage: G = Zmod(31)
sage: print(len([n for n in G if n.is_primitive_root()]))
8
```

✓

6 a.)

```
sage: p=307
sage: q=311
sage: n = p * q
```

```
sage: ef_n = (p-1)*(q-1)
sage: e = 247
sage: d = inverse_mod(e, ef_n)
sage: print(f"Private key is: ({d},{n})"
....:
```

```
....: )
Private key is: (55303,95477)
```

b.)

```
sage: c = [94755, 87565, 41862, 49231, 34234, 17479, 26771, 87503]
sage: m = []
sage: G = Zmod(n)
....: for c_i in c:
....:     m.append(G(c_i^d))
....:
sage: print(m)
[19070, 40013, 18220, 41708, 18051, 41719, 24192, 21426]
```

19 = T
07 = H
04 = E
00 = A
13 = N
18 = S
22 = W
04 = E

17 = R
08 = I
18 = S
05 = F
14 = O
17 = R
19 = T
24 = Y

19 = T
22 = W
14 = O
26 = !

THE ANSWER IS FORTY TWO!

c) One example is

e.g. One example is an e that is not coprime to φ(n) - for example 2

This means there would be no inverse to e, so a private key cannot be generated

Any concrete examples?

d.)

```
sage: def RSA(p, q, m):
....:     n = p * q
....:     phi = (p - 1) * (q - 1)
....:     e = phi.coprime_integers(phi)[-2]
....:     d = inverse_mod(e, phi)
....:     c = power_mod(m, e, n)
....:     public_key = (e, n)
....:     secret_key = (d, n)
....:     return public_key, secret_key, c
....:
sage: p = 307
sage: q = 311
sage: m = 94755
sage: print(RSA(p,q, m))
((94853, 95477), (67757, 95477), 10815)
```

e n d n

Use euler_phi(n) instead of hard calculating phi

Where is the decryptor function?

your d should be 55303 if e = 247.

(To verify if your code is correct, hard code e = 247 to get the identical answer from 6a), but this code looks good.