4CR3 A4

1.

a.

If g is not a generator of the group, we know from Lagrange's theorem that its order, o, must be some number such that: p - 1 = q * o, for some prime factor q. Therefore, when checking by raising g to the (p -1)/q, g's order will be eventually checked, leading to $g^o = 1$ (mod p). Therefore, if g is not a generator, the algorithm will return false.

If g is a generator of the group, the order of g, o = p - 1. We know that for all prime factors q, p-1/q < p - 1. Therefore, $g^{(p-1)/q}$ will not equal $g^o = g^{p-1}$, meaning $g^{(p-1)/q} \neq 1$ (mod p). Thus, the algorithm will return true. Therefore, this algorithm only returns true if and only if g is a generator.

b.

Binary exponentiation uses log(e) multiplications for an exponent e. The exponent in each iteration is (p-1)/q, which is less than p. The algorithm does binary exponentiation k times, once for each prime factor. Therefore, the complexity of the algorithm in terms of multiplications should be O(k log(p)).

c.

The algorithm returns False for the given inputs. g is a generator for a subgroup when raised to 29966994309330956238928254908 4.

2.

a.

```
for i in [0, … n-1]:
    term = $h^{3^\wedge(n-i-1)}$
    for y in [0, 1, 2]:
        if $(g^{3^\wedge(n-i-1)})^y$ = term:
            $x_i$ = y
    h = h * $g^{-x_i}$
    g = $g^3$

return $x_k * 3^k + … x_1 * 3^1 + x_0$
```

b.

The above algorithm has a run time of $O(n^2)$. The outer loop runs n-1 times, and each iteration has 2 lines with exponentiation. The first line, $h^{3^\wedge(n-i-1)}$, will take $\log(3^{(n-i-1)})$ calculations, which equals (n-i-1) * log(3) calculations, which is O(n-i). The second line $(g^{3^\wedge(n-i-1)})^y$ takes $\log(3 * 3^{(n-i-1)})$ calculations which is also O(n-i), so each iteration is O(n-i). Since these run for n-1 iterations, the final time complexity is $O(n^2)$.

c.

Using the above algorithm, x was found to be 2621519338154903134624454481960 for the given inputs.

3.
a.
The new algorithms if we were to only use a hash function H.

Gen
Choose primes p, q
Compute n = pq
Compute $\varphi(n) = (p-1)(q-1)$
Choose public exponent e
Compute $d = e^{-1} \pmod{\varphi(n)}$
We have generated public key (n, e)

Sig
Compute h = H(x)
$s = h^d \pmod n$

Ver
Compute h = H(x)
$y = s^e \pmod n$
$b = (y \overset{?}{=} h)$
accept if b = 1, reject if b = 0

b.
In the old existential forgery attack, the attacker chooses some signature s and computes $x = s^e$ (mod n). This is valid since $x = s^e$ (mod n). However, in the new RSA using just a hash function, the attacker must now find some message x such that $H(x) = s^e$ (mod n). By definition, the hash function H must be preimage resistant, meaning it is hard to find such an x when choosing s. Therefore, the existential forgery attack is not applicable to the new scheme.

c.
One major benefit that EMSA has over using only H is that EMSA is probabilistic, so signing the same message can output multiple different signatures. Using only H is deterministic, meaning it is vulnerable to attacks that can exploit this fact. Another drawback of using only hashing is that EMSA makes it so that every message is the same size, preventing attacks that use the message length and structure. Using only hashing does not prevent this. An example of an attack the hashing scheme is vulnerable to is the chosen message attack, where the attacker can try and encrypt from a list of known messages and compare them to the ciphertext.