RSA

Step 1. Define Function for Finding Inverse Number

Finding inverse number (in this case the one that q mod p = 1) is achieved by checking if xq mod p is equal to 1. This is achieved by incrementing x from 0 until such number is generated.

Python implementation:

```
def q_inv(q, p):
    x = 0
    # Do while xq%p is not equal to 1:
    while x * q % p != 1:
        # Increment x by one:
        x += 1
    return x
```

Step 2. Define Chinese Remainder Function

Variables p and q are prime numbers used for key generation

$d_p = d \bmod (p-1)$

$d_q = d \bmod (q-1)$

$q_{inv} = q^{-1} \bmod p$

These values allow the recipient to compute exponentiation $m = c^d \bmod (pq)$ efficiently as follows:

$m_1 = c^{dp} \bmod p$

$m_2 = c^{dq} \bmod q$

$h = q_{inv}(m_1 - m_2) \bmod p$

Final result (m) is: $m = m_2 + hq$

Python implementation:

```
def chinese_remainder(self, c, p, q, d):

    dp = d % (p - 1)
    dq = d % (q - 1)

    qinv = self.q_inv(q, p)

    m1 = self.pow_es(c, dp) % p
    m2 = self.pow_es(c, dq) % q

    h = qinv * (m1 - m2) % p

    return m2 + h * q
```

Note: the function *pow_es* can be replaced by regular *pow* function in Python. The function *pow_es* is custom function used to demonstrate how to implement exponentiation by squaring which is another method for improving RSA calculations and in this case used for educational purposes and not as necessity.

Defining Exponentiation by Squaring Function

This method is based on the observation, that for a positive integer n we have:

$$x^n = \begin{cases} x(x^2)^{\frac{n-1}{2}}, & if\ n\ is\ odd \\ (x^2)^{\frac{n}{2}}, & if\ n\ is\ even \end{cases}$$

In Python this can be achieved using recursive function, which will keep on calling itself, until the power is equal to 1.

<u>Python implementation:</u>

```python
def pow_es(self, base, power):
    # Raise to power by using exponentiation by squaring
    # If power is equal to 1...
    if power == 1:
        # return base value (stop recursion cycle)
        return base
    # If power is even...
    if power % 2 == 0:
        # call same function with squared base half power
        return self.pow_es(base * base, power / 2)
    # If power is odd...
    else:
        # multiply base by value of same function with
        # squared base and half of (power - 1) as arguments
        return base * self.pow_es(base * base, (power - 1) / 2)
```