Q1

- By $a \ mod \ b$, let $r = a \ mod \ b = a - bk$ for some integer k, r is remainder where $r \in \{0, \dots, b-1\}$.

First we prove the greatest common divisor of a and b can divide the greatest common divisor of b and $a \ mod \ b$.

- Let $d = gcd(a, b)$. It means $d|a, d|b$.
- So $d|a - dk$.
- So $d|gcd(b, a \ mod \ b)$.

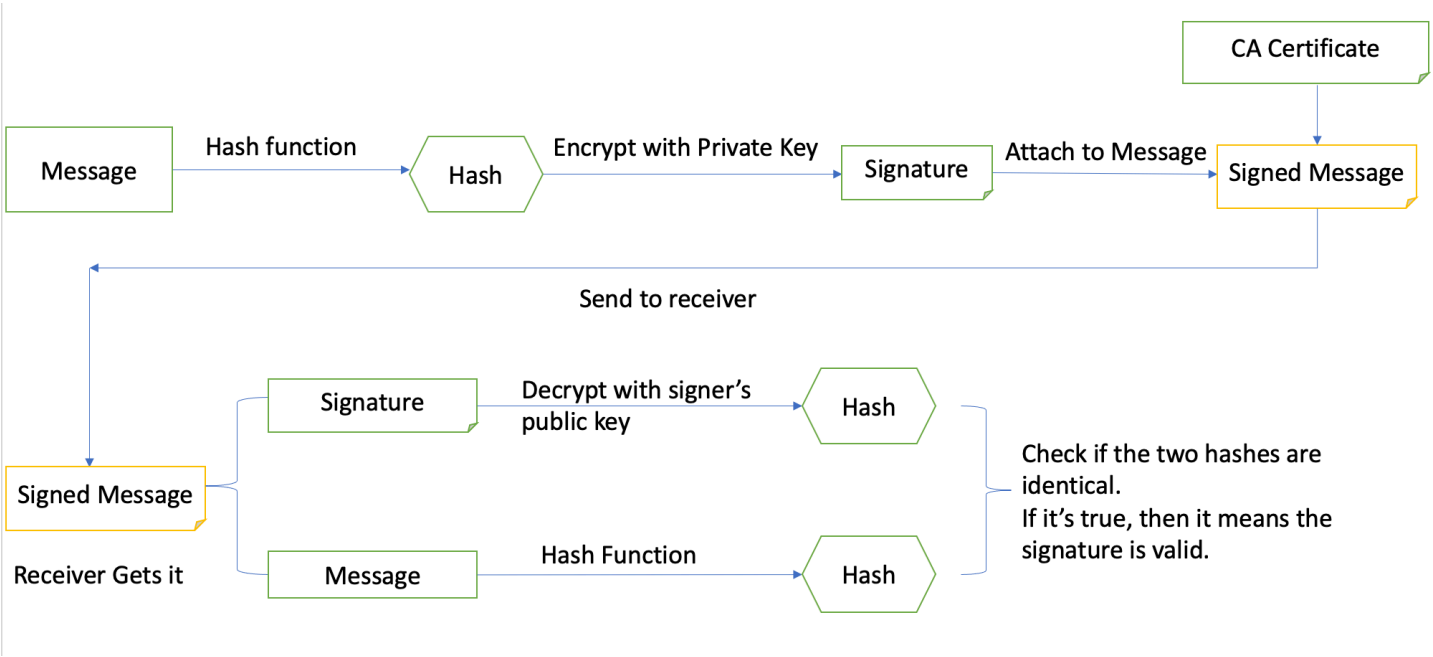
Similarly we can prove that the greatest common divisor of b and $a \ mod \ b$ is able to divide the greatest common divisor of a and b .

- Let $d' = gcd(b, a \ mod \ b)$, it means $d'|b, d'|a - bk$.
- So $d'|a$,
- So $d'|gcd(a, b)$

Therefore, $gcd(a, b) = gcd(b, a \ mod \ b)$.

Q2

- RSA signature scheme and authentication works.
- RSA signature scheme



- How signature verification works:
  - The sender has two kinds of keys, public key p_k and private key s_k. p_k would be available to everyone. A trustworthy certificate authority (CA) would mentain the public key. s_k has to be kept secretly only known to sender itself.
  - When sender wants to send a message m to any receivers, it needs to first output a singature σ

using its s_k. Then sender sends (m, σ) to the receiver.
    - When the reciever gets the m associated with signature σ, it would use the verification algorithm $Vrfy$ to determine if the signature is valid. $Vrfy$ takes as input the public key p_k, the message m, and the associated signature σ. It outputs a bit b, with b = 1 meaning valid and b = 0 meaning invalid. If it's valid, then receiver accpets the message.

Q3

Code implementation

The RSA implementation is able to encrypt and decrypt standard keyboard characters, including letters, numbers, and symbols.

In this RSA program, I showed the effectiveness with randomly generated p and q which are prime numbers with 50 bits. The program displays that its principle is implementated in the right way by successfully running on `short-test.txt` which contains letters, numbers, and symbols.

Reference: RSA (cryptosystem) - Wikipedia

(a)

What I learned

I learnt the detailes of each step in RSA algorithm and implemented the modular theories which significantly improved my understanding.

Difficulties:

- The prime_test is extremely time-consuming for large prime which is larger than 2^{64} so I have to use shorter bits. That's the reason why p and q are with 50 bits.
- It's hard to design a great strategy to convert text to a small integer. My `text2integer` strategy would got a pretty long integer which forces the n be extremely long so as to encrypt `RSA-test.txt`. The implemented strategy is inspired by rosettacode-RSA-wiki.

(b)

Source coding:

- Text -> hex-text by `hex_data = binascii.hexlify(message.encode())`
- Hex-text -> integer by `plaintext_integer = int(hex_data, 16)`

Decoding:

- Decrypt integer by `decrypted_integer = pow(ciphertext_integer, d, n)`
- Get plaintext from decrypted integer by `binascii.unhexlify(hex(decrypted_integer)[2:]).decode()`