Question 1

a)
```
def sign(M, n, d):
    s = 1
    while d > 0:
      if d % 2 == 1:
        s = (s * M) % n

      d = d // 2
      M = (M * M) % n

    return s
```

b)
```
def verify(S, n, e, M):
    return sign(S, n, e) == M
```

c)
```
def blindsign(M, x, n, e, d):
    Mb = ((x**e)* M) % n
    return sign(Mb, n, d)
```

Question 2

Since Alice is signing with her public key in both approaches, Bob cannot use Alice's private key to verify signature. Hence, the only way Bob can verify that the message came from her is by going through the process of signing the appropriate document with Alice's public key, K_A (that he already has), to compare it to the signature that he receives. Thus, to trick Bob into thinking that the message came from the attacker without actually changing the message is to swap the signature with a signature, say S_T, created using the attacker's public key, say K_T.

In Approach *A*, since *C* is encrypted using Bob's public key, K_B, it needs to be decrypted using his private key, which the attacker would not have. Thus, there is no feasible way that the attacker can decrypt *C* to get the appended signature, *S*, that was appended to *M*. If the attacker were to change anything, Bob would not be able to decrypt it properly, thus making *M* unreadable and *S* unverifiable, defeating the purpose of tricking Bob into thinking *M* came from someone other than Alice.

Hence, the active attacker would have to modify the contents sent in Approach *B*. In particular, the attacker would not need to decrypt anything and would just need to intercept *C* and *S*, swap *S* out for S_T that they then sign using K_T and send Bob both *C* and S_T. This way, when Bob signs *C* using K_A he would be getting a different value than S_T (which is what he got). Hence, Bob would think both *C* and thus *M* came from an attacker when in reality it came from Alice. With this, if Bob also had K_T and Bob signs *C* with it, he would find the output to be similar to the signature he received, S_T, and would thus think that *C*, and consequently *M*, came from the attacker.

Question 3

a) Satisfied.
The message, no matter the input size, will be divided into blocks of a predefined fixed size. This means that the entire message either has to be a multiple of the block size or padding would need to be added if a message chunk does not meet the size so that the block will follow the predefined fixed size so that all blocks are of the same size.

b) Satisfied.
Since all blocks will share a predefined fixed size, the constrain given by the modulo operation and the XOR operation will ensure we get a fixed output size.

c) Partially satisfied.
Hashing fixed sized blocks would be easier than hashing the entire message, and the XOR operation is an efficient and easy computation, but we will still have to compute the exponent, which may or may not be easy depending on the block size.

d) Not satisfied.
Depending on the hash output, an attacker may be able to find a preimage that results in the same hash output.

For example, if the initial message M consisted of an even number of blocks that are all the same, it would result in a hash output of all 0s. An attacker could then choose any message that follows this characteristic to get the same hash output.

e) Not satisfied.
Given M and $H(M)$, it would be easy for the attacker to find $M' \neq M$ such that $H(M) == H(M')$.

For example, two messages that consists of the same blocks but in different orders would result in the same hash output. E.g.:

Let M be $M_1 \,||\, M_2 \,||\, M_3$ and M' be $M_3 \,||\, M_2 || \, M_1$,
$H(M) = H(M_1, M_2, M_3)$
$= (M_1^e \bmod n) \oplus (M_2^e \bmod n) \oplus (M_3^e \bmod n)$
$= (M_3^e \bmod n) \oplus (M_2^e \bmod n) \oplus (M_1^e \bmod n)$
$= H(M')$

This is because the XOR operation is both commutative and associative. Hence, it would be easy for an attacker to find M' such that $H(M) == H(M')$

f) Not satisfied.
Such as the above example in (e), an attacker would be able to easily find M and M' such that their hash functions output the same value. Also, since it is not Second Preimage Resistant, it cannot be collision resistant.