Problem 1:

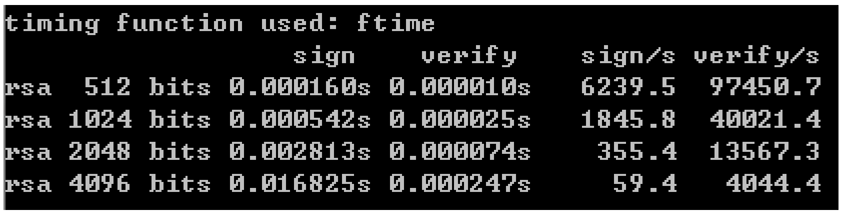
1. If we use symmetric key for authentication, then we would have the following problems:

<i> If we use the authentication based on symmetric key, all the receivers and source would have the same symmetric key. The receiver cannot identify whether the packets received came from source or other receivers.
The symmetric key authentication violates the requirements of verifying the source.

<ii>An adversary (who is also a receiver of the system) could launch a replay attack using the past packets that he received to flood the network.

2. Transmitter speed factors:

a. How many packets can the transmitter send per second using an RSA key of size 512, 1024, 2048, and 4096 bits keys?



```
timing function used: ftime
                  sign    verify    sign/s verify/s
rsa  512 bits 0.000160s 0.000010s   6239.5  97450.7
rsa 1024 bits 0.000542s 0.000025s   1845.8  40021.4
rsa 2048 bits 0.002813s 0.000074s    355.4  13567.3
rsa 4096 bits 0.016825s 0.000247s     59.4   4044.4
```

By using “openssl speed rsa”, we got the speed of signing and verifying the messages.
The transmitter can send
6239.5 packets by RSA key size of 512
1845.8 packets by RSA key size of 1024
355.4 packets by RSA key size of 2048
59.4 packets by RSA key size of 4096

b. How many packets can the receiver verify per second for RSA key size of 512, 1024, 2048, 4096 bits keys?
The receiver can verify
97450.7 packets by RSA key size of 512
40021.4 packets by RSA key size of 1024
13567.3 packets by RSA key size of 2048
4044.4 packets by RSA key size of 4096

c. Why is the signing time different from the verification time? Would the length of the packet matter very much in the signature/verification time?
Because the signing part is using the private key d and the verification part is using the public key e. By computing several pairs of e and d, we could see that the private key is significantly bigger than e, so doing M^d cost much longer time than M^e, so signing part is taking much more time than verification.
The length of the packet doesn’t matter much in the both the signing and verification because we sign and verify the hash of the packet.

d. Indicate the specs of the machine you used to run openssl (e.g., lab VM, or Quadcore i7 laptop 2.4GHz with 8GB RAM, etc.)
Core i5 2.4Hz, 16GB RAM.

3. If the sender node does not have the capability to sign all the packets individually using its private key, propose a technique to amortize the signatures and trades-off the computation with a delay in verification. Discuss potential DoS attacks on your approach.

The source could have a buffer to buffer a certain number of packets. The source store the packets into the buffer when it cannot deal with the packets. Once the buffer is filled up, the source group them by distinct receiver, and sign each group once before sending the following packets.

In this way, the source could reduce the number of signing.

Potential DoS attacks:

The adversary could send huge amounts of packets to the source to occupy the buffer, thus leaving no space for the normal packets to be signed and answered until the previous filled buffer is answered.

Problem 2:

This protocol is susceptible to reflection attack.

Session 1:

T $\rightarrow$ B: I am A,R1

B $\rightarrow$ T: R2,$K_{AB}\{A\}$,$K_{AB}\{B\}$,$K_{AB}\{R1\}$

We need $K_{AB}\{R2\}$ to complete the attack, so we ask B to compute for us.

T $\rightarrow$ B: I am A, R2

B $\rightarrow$ T: R3, $K_{AB}\{A\}$,$K_{AB}\{B\}$,$K_{AB}\{R2\}$

Now we have $K_{AB}\{R2\}$, so we can finish session one

T $\rightarrow$ B: $K_{AB}\{B\}$,$K_{AB}\{A\}$,$K_{AB}\{R2\}$

Besides, this protocol is susceptible to password guessing attack.

T $\rightarrow$ B: I am A, R1

B $\rightarrow$ T: R2, $K_{AB}\{A\}$, $K_{AB}\{B\}$, $K_{AB}\{R1\}$

T can go offline password guessing attack by using pair <R1, $K_{AB}\{R1\}$>

To solve these two kinds of attacks, we can modify the protocol as follows:

A $\rightarrow$ B: I am A

B $\rightarrow$ A: R1

A $\rightarrow$ B: R2, $K_{AB}\{A\}$, $K_{AB}\{B\}$, $K_{AB}\{R1\}$

B $\rightarrow$ A: $K_{AB}\{A\}$, $K_{AB}\{B\}$, $K_{AB}\{R2\}$

In this way T could not get the pair <R1, $K_{AB}\{R1\}$> to do offline guessing.

Further, the reflection attack won't work because T doesn't know $K_{AB}\{R1\}$ to continue the authentication.

Problem 3: RSA

1. $p = 23$ $q = 17$
   $m = 23 * 17 = 391$
   $n = (23 - 1) * (17 - 1) = 352$

$e = 3$
$d * e \bmod n = 1$
$d = 235$
$d * e = 705 \bmod 352 = 1$

$\therefore$ $d = 235$

2. $M = 4$

$\therefore E(M) = E(4) = 4^e \bmod m = 4^3 \bmod 391 = 64$

3. $D(M) = D(2) = 2^d \bmod n = 2^{235} \bmod 391 = 2^{128} * 2^{64} * 2^{32} * 2^8 * 2^2 * 2^1 \bmod 391$
   $= 246$

4. Diffie-Hellman algorithm
   shared key $= g^{ab} \bmod p = 3^{5 * 2} \bmod 113 = 63$

Problem 4:

In comparison with protocol 2, the additional service that protocol 1 provided is the authentication of the source. In protocol 1, A and B are sending its key to each other by signing with its private key. While in protocol 2, A and B simply send the C1 and C2 encrypted with public keys without authentication. The protocol 2 generates the shared key without authentication of the source of message.

The required conditions to be satisfied for this additional service to be trusted are listed as follows:
1. Private keys of both A and B must be kept secret.
2. g, p and A and B's public key must be known by both A and B before the communication starts. A and B need g, p to generate the shared key and public key of each other to verify the signature.

If authentication is not needed, the advantage of protocol 2 is that A and B don't need to agree on the value g and p to start the communication. Furthermore, the computation would be much easier and faster by doing $\oplus$ instead than exponential arithmetic.

Problem 5:

This protocol is vulnerable to breaching the database of B.
After breaching the database of B, we would have the value b and g^W mod p.
We could calculate the g^{Wb} mod p by computing $(g^W \bmod p)^b$ mod p.
Then we eavesdrop the message from A to B: g^a mod p
We could calculate the g^{ab} mod p by computing $(g^a \bmod p)^b$ mod p.
Then we could compute the hash of Hash(g^{ab}mod p,g^{Wb}mod p)
Since the W is weak, we could try different W' to get the secret W by:
Hash(g^{ab}mod p,$g^{W'b}$mod p) = Hash(g^{ab}mod p,g^{Wb}mod p)
Then the adversary has the secret W to continue the attack against A.

Problem 6:

This protocol is vulnerable to eavesdrop. Since A is sending clear challenge R1 to B and Hash((K+R1) mod 2^{64}), R2 to authenticate itself. Any adversary can record the challenge R1 and the corresponding Hash to complete the authentication. The whole attack process goes as follows:
T→B: I am A
B→T: R1
T→B: Hash((K+R1) mod 2^{64}), R2

The last message of B→ A is not considered for authenticating A to B.

The adversary records the R1 and Hash((K+R1) mod 2^{64}) and send it when B sends R1 the next time for authentication. Besides, the adversary could record all pairs of <Rn, Hash((K+Rn) mod 2^{64})> for any other challenge.

Problem 7:

The usage of S-Hash and C-Hash help to eliminate the danger of being attacked by middle-man-attack which is dangerous for Diffie-Hellman algorithm. And the encryption part is using the DH key to do AES-CBS encryption, which proved to be safe to use.

The only flaw would be compromising one of the server or client in this system. After compromising the client/ host, the only thing that the adversary needs to do is to try to do offline guessing on the 2 parts of 4-digit pin. 10^4 possibilities would be easily broken by computers.