Exercise 6.1

The first solution fails to satisfy the security policy. Assume the attacker is User B.

When B intercept the traffic among A and D, he gets $E_{pk_D}(R)$, $S_{sk_A}(E_{pk_D}(R))$, A. Then he can use his own key sk_B to sign the first part of this message $E_{pk_D}(R)$, and use it to replace the original signature. Also he need to replace the sender to B. So what he send to D is $E_{pk_D}(R)$, $S_{sk_B}(E_{pk_D}(R))$, B. When D receive this message, he will use B's public key to verify the signature, and then send back result to B.

This breaks the second security policy. In this situation, B can get information about A asked for.

Exercise 5.11

> *We implement all the test cases in "CryptoModule/main.go" including Exercise 5.11 and 6.10*

1. we create a pair of keys with 2000 bits length using `RSA.KeyGen` function.
2. Then we randomly generate a number as plaintext.
3. We use `RSA.Encrypt` function to encrypt it with public key.
4. Then we use `AES.EncryptToFile` and `AES.DecryptFromFile` to encrypt the RSA priavte key and then load and decrypt it from file. The key used for AES is randomly generated.
5. At last, we use decrypted private key to decrypt ciphertext, and we can see that the message is same as original plaintext.

Exercise 6.10

1. We create 2 messages,message1 "hello world" and message2 "goodbye world", first we use `Hash()` function to get the hash value of message1 and then use `Sign()` function to get the signature of the hash value. In `Verify()` function,

public key is used to decypt the signature, if this value is same as the hash value of message then it can be verified and returns *true*, otherwise, *false*.

2. We randomly generate a 10KB string, and measure the time spent on hashing it. We do it 10 times and the average result is about `37.135µs` per message thus `4.53e-4µs` per bit.
3. We also test 10 times and the average time spent on producing an RSA signature on a hash value when using a 2000-bit RSA key is `5.68ms`.
4. We create a plaintext with size of `2.4MB`.
   Time used to hash and sign this hash value is `17.70ms` while time used to sign the plaintext is `36.93ms`. Therefore, when the size of plaintext is large enough, hashing makes signing more efficient.