- تمرینات زیر را از مرجع اصلی درس؛ کتاب **Understanding Cryptography** حل کنید.
  ۱) سوال ۱۲.۳
  ۲) سوال ۱۲.۵
  ۳) سوال ۱۳.۹
  ۴) سوال ۱۳.۱۱
  ۵) سوال ۱۳.۱۵

6- Write a client server program in your favorite programming language, where the client and the server can send arbitrary messages with at most 4096bit length. Use cryptographic mechanisms to guarantee the integrity and confidentiality of exchanged messages over the insecure communication channel. Take following points into account:

a) The adversary can read the source codes of client and server. Additionally, he can access and read files on the system. So, you shall not hardcode or save any of your secrets within your codes or in any of the files on your system.

b) The attacker can eavesdrop the channel. Therefore, to ensure the confidentiality of messages, you might want to use a mechanism with which the sender converts a message into an alternative form unreadable by the attacker using a shared secret, which is only known to authorized parties, and the receiver reverts the message to its original form using the same secret. Take care about the strength of the mechanism you choose and pay attention to weaknesses such as determinism and malleability.

c) As the channel is not initially secure and neither can you hardcode the shared secret, you shall use a mechanism through which one of the parties chooses the secret and sends it to the other party in a secure manner. Here, to guarantee the confidentiality of the exchanged secret, you should opt a cryptographic mechanism with which the sending party converts the secret, itself, to a form, again, obscure to the adversary. But, this time, the only person who can convert the secret to its original form is the receiving party who has a private secret, which helps it about returning the shared secret to its initial form.

d) Finally, besides eavesdropping, the attacker can actively modify the packets exchanged between parties, even if he doesn't have the slightest idea about their contents. To prevent this from happening and guarantee the integrity of the messages, you should attach to the message block, a unique value derived from the message, itself. This value should be the output of a cryptographic function that receives the message as input. It should, further, have this property that if the adversary changes any of the original message's bits the value associated with the new message should completely differ from the attached value. So that, this way, the other side can recognize the message has been modified on the way. Moreover, it should be computationally infeasible for the attacker to guess the content of the message from this attached value, or find a message different from the original message that, if given to the same cryptographic function as input, results in the same value.

For simplicity, assume that capabilities of the attacker are limited to those mentioned above, and he cannot do anything further, such as impersonating communicating parties, and so on.

Optional

7- Using SHA256, implement the HMAC algorithm in your favorite programming language. You should implement all parts except the hash algorithm on your own. Compare your code's final results with those of built-in HMAC implementations.

8- Do the following exercises surrounding Davies–Meyer algorithm;

a) Implement this compression function in your favorite programming language. Your code should receive a string of arbitrary length and compute its Davies–Meyer hash.

b) Using the code you've written in previous part and a random key, calculate the secret prefix Message Authentication Code of an arbitrary input string.

c) Conduct an attack against the previously calculated secret prefix MAC. Note that your original message's length in addition to key length should be a multiple of the compression function's block size (the number of letters in the message should be a multiple of 16).

d) Check whether your attack in previous part was successful by appending the original message string to the additional string in the previous part, computing the hash of the resulting string and comparing the resultant with the hash in previous part.