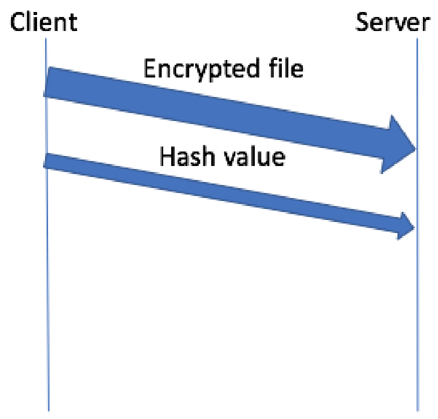
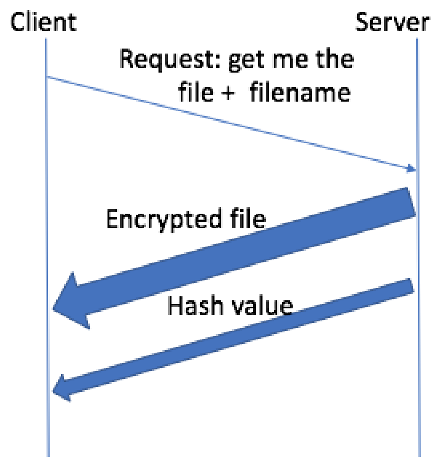
ESET 415 – Fall 2020 – Client-server programming + security using RSA

Program a secure network application using socket programming using Python as the programming language. This network application should provide confidentiality and integrity.
(Hint: This is similar to the labs we have been doing, as it combines the encryption functions we have been doing and hash values)

Details:
- Develop a client and a server application to transfer encrypted files + hash value over UDP transport protocol, as in Figure 1.





Let's break this project into simple parts:

1) File Transfer - Modify your UDP client/server sockets program to do the following:		Total points
client requests to **upload** (store) a file (of any size and type) in the server ("put file" command), and client sends the file to the server. Server saves the file in its local directory. (To verify this, encryption is disabled.)	Fully working with sample files - 10 Almost correct – 7 Partially correct – missing main parts – 3 Not submitted - 0	
client requests to **download** a file (of any size and type) from the server ("get file" command), and server sends the file. Client saves the file in local directory. (To verify this, encryption is disabled.)	Fully working with sample file, between different hosts - 10 Almost correct – 7 Partially correct – missing main parts – 3 Not submitted - 0	

2) **Integrity:** provide means for the receiver to verify that the file has not been modified in transit. A 256-bit hash value, generated using secure hash algorithm (SHA) will be sent and checked at the receiver.	Fully working with sample files - 10 Almost correct – 7 Partially correct – missing main parts – 3 Not submitted - 0	
3) **Confidentiality/Encryption:** Encrypt the file using RSA public key encryption (encrypt using the receiver's public key). It should work with multiple key pairs. This encryption should work well with the file transfer of steps 1 and 2 ("put" file, and "get" file). The client should have a public key [e1, n1], [d1,n1], where n1 = p1 *q1. The server should have a different key pair [e2,n2], [d2,n2], where n2 = p2*q2. Note n1 and n2 have to be bigger than 255 (to support ASCII and binary files).	Fully working with sample files - 10 Almost correct – 7 Partially correct – missing main parts – 3 Not submitted - 0	
4) **Decryption** of the received file using using RSA public key encryption (decrypt using the receiver's private key). It should work with multiple key pairs. This decryption should work well with the file transfer of steps 1 and 2 ("put" file, and "get" file) The client should have a public key [e1, n1], [d1,n1], where n1 = p1 *q1. The server should have a different key pair [e2,n2], [d2,n2], where n2 = p2*q2. Note n1 and n2 have to be bigger than 255 (to support ASCII and binary files)	Fully working with sample files - 10 Almost correct – 7 Partially correct – missing main parts – 3 Not submitted - 0	

** Reference on RSA algorithm: our class notes, and **Section 19.3 of Applied Cryptography – pages pp.466-474**
You should support all types of files (e.g., text files, such as .txt, .json, .csv) and binary files (e.g., pdf files, images). Sample test files will be provided on ecampus. You can demonstrate this project using your local computer (one client and one server program running in the same local host). This work should be done individually.
Due dates:

- **Demonstrations – should be completed by May 4 at the end of the day (Monday night)**