Report for Problem Set 2:

The problem statement expected us to build an application that could be used to encrypt and sign a file to be sent by email. It also expected that the application be efficient in terms of speed of execution.

The strategy used to tackle this problem is commonly known as Pretty Good Privacy(PGP).
PGP is an encryption program that ensures cryptographic privacy and authentication for data communication.

Both symmetric and asymmertic encryption algorithms along with a signing alogorithm have been used to implemet the PGP strategy and build the application.

Symmetric encryption algorithm : AES-GCM
Asymmetric encryption algorithm : RSA
Signing algorithm : RSA

Steps taken to implement the applications:-

Encrypting:-

1. We generate a random 32 byte key and a random 16 byte initialization vector at runtime. This key and initialization vector will be used with the symmetric encryption algorithm (AES-GCM) to encrypt the data present in the input file.

-Keys are used to control the operation of a cipher so that only the correct key can convert the encrypted file into its respective format. The size of the key is directly proportional to the security of the algorithm. A key with more bytes is harder to crack (especially by using brute-force). A key of 256-bit length is many times harder to guess as compared to a key of 128-bit length. Larger keys may increase the computation time. But it wont lead to a very heafty increase in computation time, that would reflect on the efficiency of our application.
So, in this case to make it hard for any adversary to find our key, we create a random 256 bit key.
In case of the initialization vector we could use one with lager size, but as long as we are not using the same iv more than once, it should not be a problem if we use a 128 bit iv.

2. Use the generated key and iv to symmetricaly encrypt (AES-GCM) the message that is to be sent to the receiver on the other end.

-We use symmetric encryption to encrypt the large chunks of data as it is much faster compared to the asymmetric algorithms (alomost 1000 times faster). AES alone is not sufficient to ensure that our application is providing cryptographic privacy. So we use it with an operation mode called Galois/Counter Mode (GCM). Use of GCM mode of operation provides high speed communication with reasonable hardware resources. It is designed to provide both data authenticity and confidentiality.

-GCM mdoe applied to AES converts AES form a block-cipher to AES-GCM which is a stream cipher. So we dont need to worry about padding our input to corrospond with the block size.

3. Now we use the destination public key provided as input to encrypt the 32 byte key we generated at runtime using an asymmetric encription algorithm (RSA).

-The public key used as input was generated using OpenSSL and had a length of 2048 bits. We could choose a much larger key size but in case of RSA with every doubling of the RSA key length, decryption becomes 6-7 times slower. And choosing keys of smaller size make them vulnerable to brute-force attacks.

-Compared to symmetric algorithms, asymmetric ones are much slower (almost 1000 times slower). But symmetric algorithms need a secure channel to operate upon where as asymmetric algorithms dont.
So the strategy used here, is to encrypt the 256 bit key generated at runtime using the asymmetric encription alogorithm (RSA) and send the encrypted key. This will make our application more efficient both in terms of speed and security.

4. Sign the encrypted message using RSA.

-Apart form encryption RSA can also be used as a signing algorithm. It creates a hash using the input message by using a hashing function. Here we have used the SHA256 hashing algorithm as it is a one-way function, deterministic, fast to compute, collision resistant and is resilient to pre-image attacks. After the hash is created RSA signs the hash using the sender's private key.

5. All these encrypted messages are put together and sent to the destination where they are decrypted.

Decrypting:-

1. Once the message is received we need to decrypt it in order to access its contents. The first thing to do is check the signature on the message and to make sure that it arrived from the user who claims to have sent it. We can do this by verifying the signature using RSA and the sender's public key.

-The second part to using RSA for signing, is that we can use the public key of the sender to verify whether the message was signed by the sender. This is done by using the sender's public key. If the adversary tries to make any changes to the message while it is in transit, this verification will fail and the receiver would know that this is not the original message.

2. Now we need to use RSA again, but this time to retrieve the key which we generated at runtime and encrypt the message.

-We use RSA along with the receiver's private key to decrypt the encrypted key and retrieve the 32 byte key that we generated at runtime. This is the symmetric key that was used to encrypt the message.

3. Now that we have the symmetric key used to encrypt the data, we can use it to retrieve the original message and write the data to a file.

-This data will be exactly same as the data of the input file which we used during the encryption process.