**Name:** Zeeshan Hamid  
**Roll No.:** 21L-1876  
**Section:** BCS 7-C

**Information Security**  
**Assignment 1**

**Question 1:**

ECB mode requires padding because it encrypts data in fixed block sizes. For DES this block size is 8 bytes. If the data isn't an exact multiple of 8 bytes then the remaining bytes need to be padded to form a complete block. Without padding, ECB wouldn't be able to process the last incomplete block & causes issues during encryption.

OFB mode does not require padding because it operates as a stream cipher. It generates a continuous key stream that is xored with the data byte by byte. Since OFB processes data as a stream it doesnot rely on fixed size blocks and can handle any length of data without the need for padding.

**Question 2:**

The problem with ECB mode is that the output reveals a lot about the plaintext as each block of data is encrypted independently using ECB and always produces the same ciphertext for identical blocks. This implies that the plaintext will show patterns such as repeated patterns like similar places in an image or text. These repeated blocks appear one after another in the ciphertext. Thus, if the data to be encrypted has discernible patterns (e.g., clear regions with the same color in an image), ECB will reveal those patterns at the ciphertext level. Therefore, for all intents and purposes, it is not a secure mode under almost any circumstance.

On the other hand, OFB (Output Feedback) mode works like a stream cipher. It generates a unique key stream for each block of data even if the plaintext blocks are similar to each other. The key stream is XORed with the plaintext,, so even if there are repeated patterns in the plaintext, the output will look random, making it much harder to detect any patterns about the original data. This is why OFB is much better at hiding information about the plaintext compared to ECB.

**Question 3:**

In the scrypt key generation, the following parameters are required:

1. **Password:**  
   This is the user-provided password used as the base for generating the cryptographic key. The password is in the form of a string that is transformed into a secure key.
2. **Salt:**  
   A randomly generated value (in my case, 16) that is added to the password before hashing. It ensures that even if two users have the same password, their keys will still be different because they have different salts.
3. **Key Length:**  
   key\_len specifies the length of the derived key. In this case, it is 8 because DES keys are 8 bytes long. It is adjustable.
4. **CPU/Memory Cost Factor - N:**  
   This defines the difficulty of the key derivation process. The larger the N, the more computationally expensive it is to derive the key, making brute-force attacks harder. In my code, it's set to 2\*\*14 (16,384). It is crucial for making scrypt resistant to hardware attacks by increasing computational and memory requirements.
5. **Block Size Factor - r:**  
   This controls memory usage. The larger the value of r, the more memory is used. In my code, it's set to 8, which is a reasonable default for most use cases.
6. **Parallelization Factor - p:**  
   This defines how many parallel computations can be done. A higher value allows the function to use multiple threads or cores during computation.