### **Q.1]** Compare stream cipher and block cipher

**Basic Operation**:

* **Stream Cipher**: Encrypts data one bit or byte at a time using a keystream generator. Each plaintext bit or byte is combined with the corresponding keystream bit or byte to produce ciphertext, typically using an XOR operation.
* **Block Cipher**: Encrypts data in fixed-size blocks (e.g., 64 bits or 128 bits). Each block of plaintext is processed independently or with feedback from previous blocks, depending on the mode of operation. The same key is used to encrypt each block, but the transformation is complex and involves multiple rounds of substitution and permutation.

**Speed**:

* **Stream Cipher**: Generally faster because it processes smaller units of data. This makes it suitable for real-time applications where data arrives continuously and needs to be processed immediately.
* **Block Cipher**: Generally slower due to the overhead of processing larger blocks. The complexity of multiple rounds of transformations in block ciphers can add latency, though modern block ciphers are optimized for performance.

**Complexity**:

* **Stream Cipher**: Usually simpler in design and implementation, often based on linear feedback shift registers (LFSRs) or similar constructs. The simplicity makes them efficient in both software and hardware implementations.
* **Block Cipher**: More complex due to the need to manage block sizes, padding, and different modes of operation (e.g., ECB, CBC, CFB, OFB, and CTR). The complexity helps achieve higher security but requires more computational resources.

**Synchronization**:

* **Stream Cipher**: Requires precise synchronization between sender and receiver. If synchronization is lost, the decryption process will fail, and the plaintext will be garbled. This sensitivity requires mechanisms to ensure synchronization is maintained.
* **Block Cipher**: Less sensitive to synchronization issues since it processes fixed-size blocks independently (in ECB mode) or with predictable patterns of dependencies (in other modes). Even if some blocks are lost or corrupted, subsequent blocks can still be decrypted correctly.

**Error Propagation**:

* **Stream Cipher**: Errors in transmission generally affect only the portion of data where the error occurs, but loss of synchronization can be problematic, leading to decryption failures for all subsequent data until synchronization is restored.
* **Block Cipher**: Errors in one block can affect subsequent blocks depending on the mode of operation. For example, in Cipher Block Chaining (CBC) mode, an error in one block will propagate to the next block, causing additional plaintext corruption.

**Applications**:

* **Stream Cipher**: Suitable for applications where data comes in continuous streams, like secure voice or video transmission, or in network protocols where data packets are processed on the fly. Examples include A5/1 for GSM encryption and RC4 for SSL/TLS (though RC4 is now considered insecure).
* **Block Cipher**: Suitable for applications where data can be processed in chunks, like file encryption, database encryption, or disk encryption. Examples include the Advanced Encryption Standard (AES) and the Data Encryption Standard (DES).

**Security**:

* **Stream Cipher**: Vulnerable to attacks if the same keystream is reused, leading to keystream reuse attacks where attackers can recover plaintexts by XORing two ciphertexts encrypted with the same keystream. Properly managed, they can be very secure.
* **Block Cipher**: Provides stronger security guarantees for large amounts of data, especially when combined with appropriate modes of operation and key management practices. Modern block ciphers like AES are considered highly secure and are widely adopted.

### **Q.2]** Discuss RC4

**Basic Operation**:

* RC4 is a stream cipher designed by Ron Rivest in 1987. It generates a pseudorandom keystream that is XORed with the plaintext to produce the ciphertext. The same process is used for decryption, as XORing the ciphertext with the same keystream recovers the plaintext.

**Key Scheduling Algorithm (KSA)**:

* RC4's KSA initializes a 256-byte state array (S) using the key. The array is then permuted based on the key, involving multiple swaps of array elements. This ensures that the initial state of the array is dependent on the key and provides the starting point for the keystream generation.

**Pseudorandom Generation Algorithm (PRGA)**:

* After the KSA, the PRGA is used to generate the keystream. It continuously shuffles the state array and produces output bytes by selecting values from the array. The selected values form the keystream, which is then XORed with the plaintext.

**Speed and Efficiency**:

* RC4 is known for its simplicity and speed, making it efficient in both software and hardware implementations. Its simple operations (e.g., swapping array elements, XOR) make it computationally lightweight, allowing for high-speed encryption and decryption.

**Applications**:

* RC4 was widely used in various applications, including SSL/TLS for securing internet traffic, WEP for Wi-Fi security, and various other protocols requiring fast encryption. However, its usage has declined due to security vulnerabilities discovered over time.

**Security Weaknesses**:

* RC4 has several well-known vulnerabilities. Key scheduling weaknesses can lead to the first few bytes of the keystream being non-random, making it susceptible to attacks like the "RC4 biases." The WEP protocol, which used RC4, was notably broken due to these weaknesses. Consequently, many organizations have phased out RC4 in favor of more secure algorithms.

**Biases in Keystream Generation**:

**Initial Keystream Biases**: The initial output bytes of the RC4 keystream exhibit non-random biases. This weakness can be exploited to recover parts of the plaintext if the same key is used to encrypt multiple messages.

**Fluhrer, Mantin, and Shamir (FMS) Attack**:

This attack exploits weaknesses in the RC4 key scheduling algorithm, particularly when RC4 is used with weak initialization vectors (IVs) as in the WEP protocol. By capturing enough packets, an attacker can recover the key used for encryption.

**Current Status and Recommendations**:

* Due to its vulnerabilities, RC4 is no longer recommended for use in new systems. Modern standards and protocols have deprecated RC4 in favor of more secure ciphers like AES. For instance, RFC 7465 prohibits the use of RC4 in all versions of TLS, emphasizing the need for stronger cryptographic practices.