RC5: is a symmetric block cipher known for its simplicity and flexibility in key and block sizes.
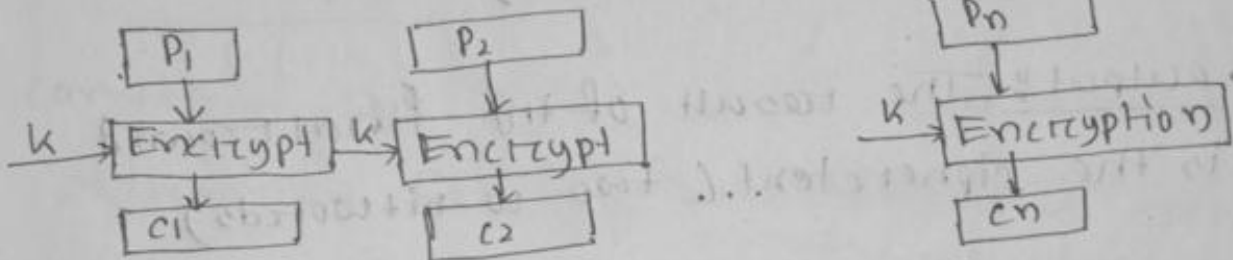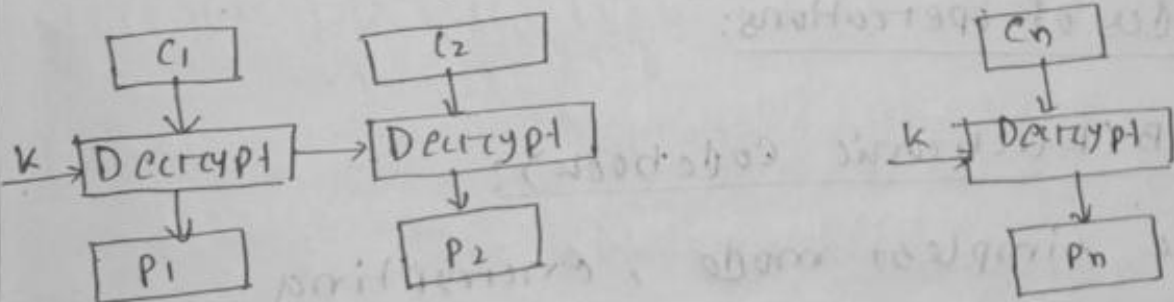
Encryption

Decryption

RC5 Block Diagram:

- **Input:** The plaintext is divided into two words (w bits each)

- **Initial Key Addition:** The input words are combined with a portion of the expanded key

- **Mixing /Circular-shift:** These operations (XOR and Rotations) are applied iteratively in multiple rounds.

- **Key Schedule:** Generates the expanded key from the secret key.

- **Output:** The result of the final round is the ciphertext (two w-bit words).

Modes of operations:

1) **ECB (Electronic Codebook):**

The simplest mode, encrypting each block independently, suitable for short, random data but can reveal patterns in non-random.

ii) **CBC (Cipher Block Chaining):**

Each block is XORed with the previous ciphertext block before encryption. Providing better security than ECB

iii) OFB (output Feedback):

Uses the previous ciphertext block to generate the keystream, making it suitable for noisy channel.

iv) CFB (cipher Feedback):

Converts the block cipher into a stream cipher by feeding back previous ciphertext blocks.

v) Counter (CTR):

Uses a counter to generate the keystream allowing for parallel encryption and decryption.

Java implementation of RC5:

```java
import java.nio.ByteBuffer;
import java.util.Arrays;

public class RC5 {

  private final int w= 32;
  private final int r = 12;
  private final int b = 16;
  private final int[] s;
  private final int pu = 0xb7e15163, qw = 0x9e3779b9;

  public RC5 (byte[] key) {

    int c = (key.length +3)/4;
    int[] L = new int[c];

    for (int i=0; i< key.length; i++) {
      L[i/4] |= ((key[i] & 0xFF) << (8 * (i%4)));}

    int t = 2*(r+1);
    s = new int[t];
    s[0] = pu;
    for (int i=1; i<t; i++) {
        s[i] = s[i-1] + qw; }
```

```java
int A=0; B=0; i=0; j=0;
int v = 3 * Math.max(c,t);
for(int s=0; s<v; s++){
A = S[i] = Integer. rotateLeft((S[i]+A+B),3);
B = L[j] = Integer. rotateLeft((L[j]+A+B),(A+B));
i = (i+1)%c;
j = (j+1)%l;
}
}

public int[] encrypt(int[] pt){
int A = pt[0]+ S[0];
int B = pt[1]+ S[1];
for (int i=1; i<=r; i++){
A= Integer. rotateLeft(A^B,B)+ S[2*i];
B= Integer. rotateLeft(B^A,A)+ S[2*i+1]; }

return new int[] {A,B}; }

public int[] decrypt(int[] ct){
int A = ct[0];
int B = ct[1];
for(i=r; i>=1; i--){
```

```java
        B = Integer.rotateRight(B-S[2*i+1], A)^A;
        A = Integer.rotateRight(A-S[2*i], B)^B; }

        A -= S[0];
        B -= S[1];

        return new int[] {A,B}; }

    public static void main(String[] args) {

        byte[] key = "MySecretKey12345".getBytes();

        RC5 rc5 = new RC5(key);

        int[] plaintext = {0x12345678, 0x9ABCDEF0};
        System.out.printf("Plaintext:%08x %08x\n",
                          Plaintext[0], Plaintext[1]);

        int[] ciphertext = rc5.encrypt(plaintext);
        System.out.printf("Encrypted: %08x %08x\n",
                          ciphertext[0], ciphertext[1]);

        int[] decrypted = rc5.decrypt(ciphertext);
        System.out.printf("Decrypted: %08x%08x\n",
                          decrypted[0], decrypted[1]); }}
```

Sample output:

```
plaintext:  12345678 9ABCDEF0
Encrypted:  E2C3456  5E9IA7B9
Decrypted:  12345678  9ABCDEF0
```