Sadia Chowdhury
IT-21062

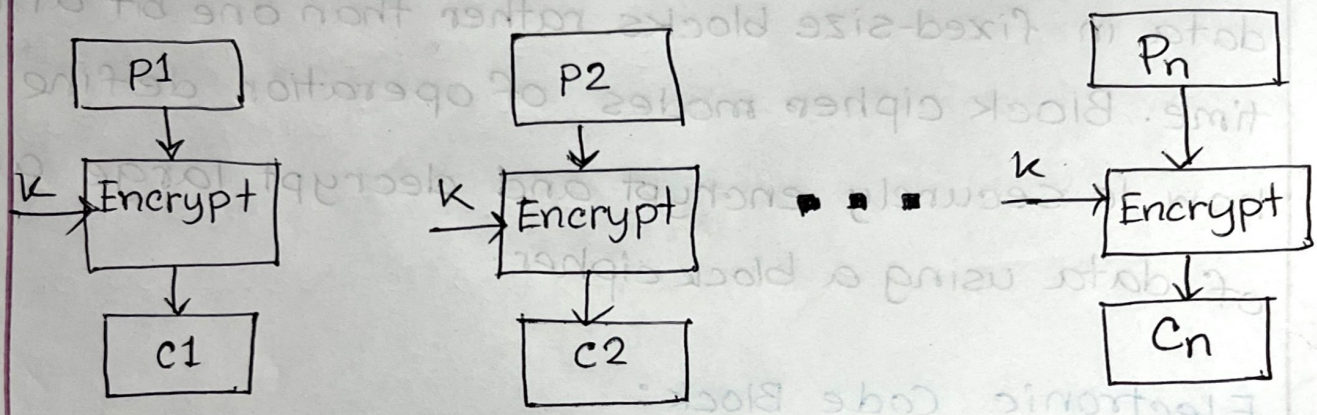
Block Cipher Modes of Operation -

A block cipher is an encryption algorithm that processes data in fixed-size blocks rather than one bit at a time. Block cipher modes of operation define how to securely encrypt and decrypt large amounts of data using a block cipher.

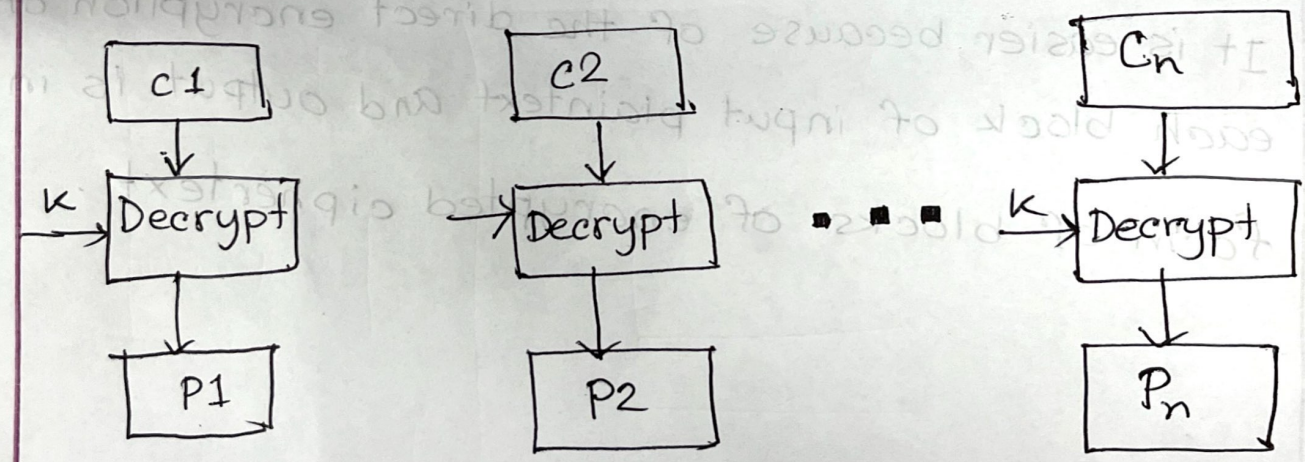
Electronic Code Block:

ECB is the easiest block cipher mode of functioning. It is easier because of the direct encryption of each block of input plaintext and output is in the form of blocks of encrypted ciphertext.

Encryption →



Decryption →

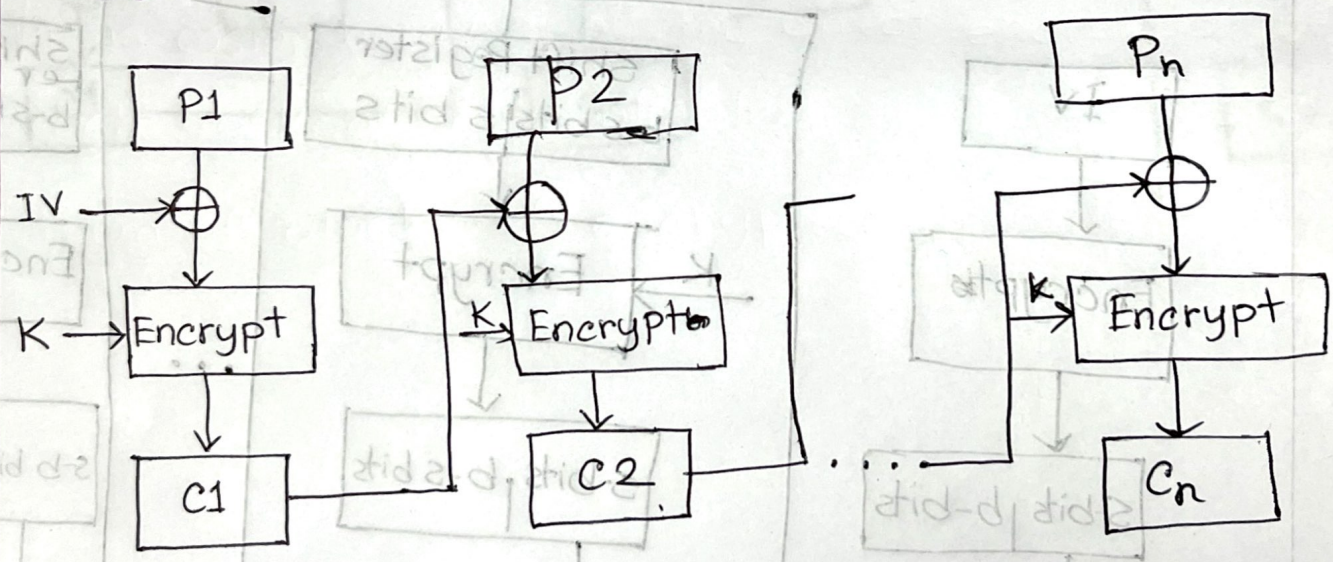


Cipher Block Chaining (CBC)

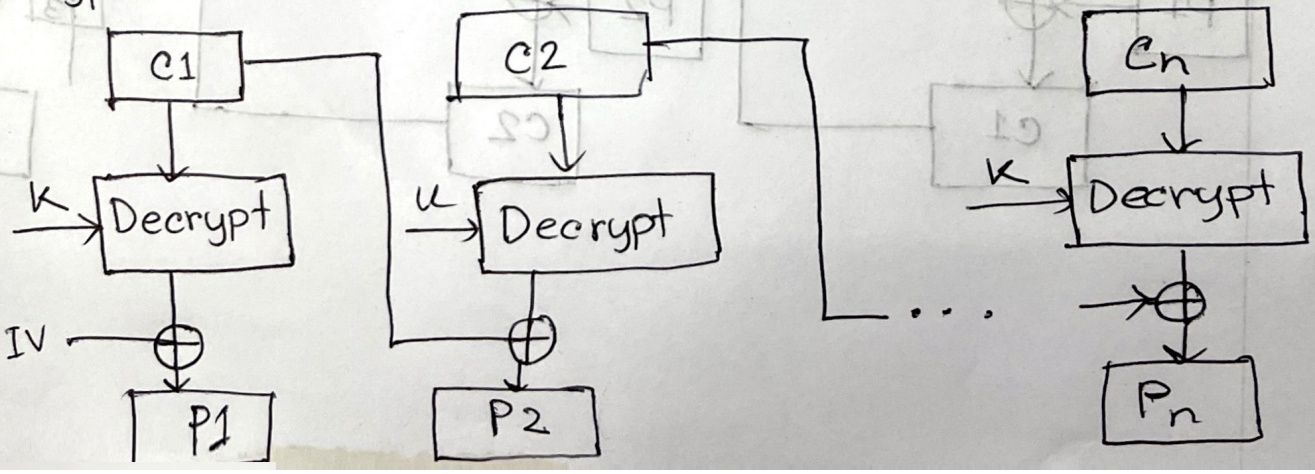
Cipher Block Chaining or CBC is an advancement made on ECB since ECB compromises some security requirements. In CBC the previous cipher block is given as input to the next encryption algorithm after XOR with the original plaintext block.

Diagram:

Encryption:



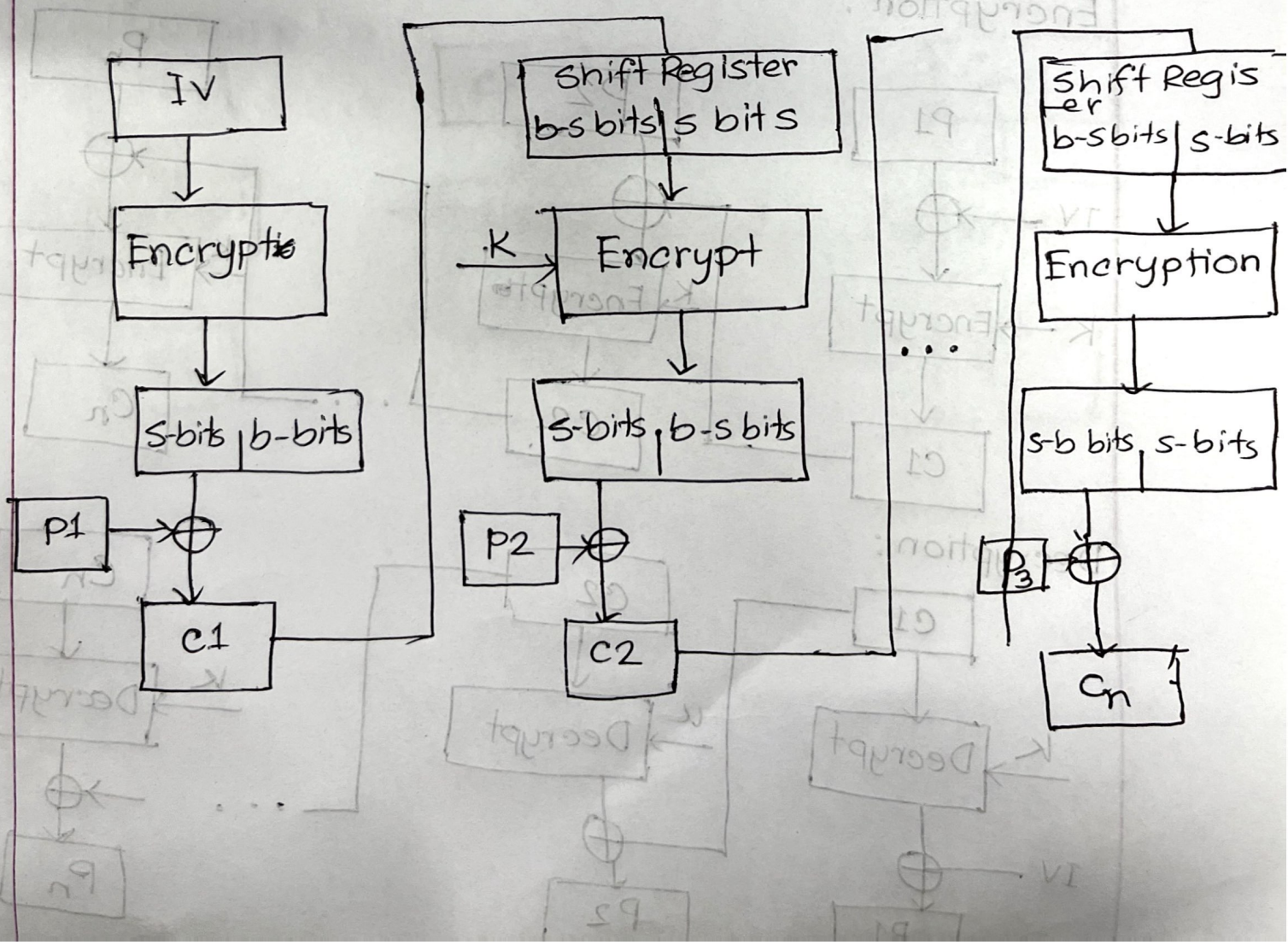
Decryption:



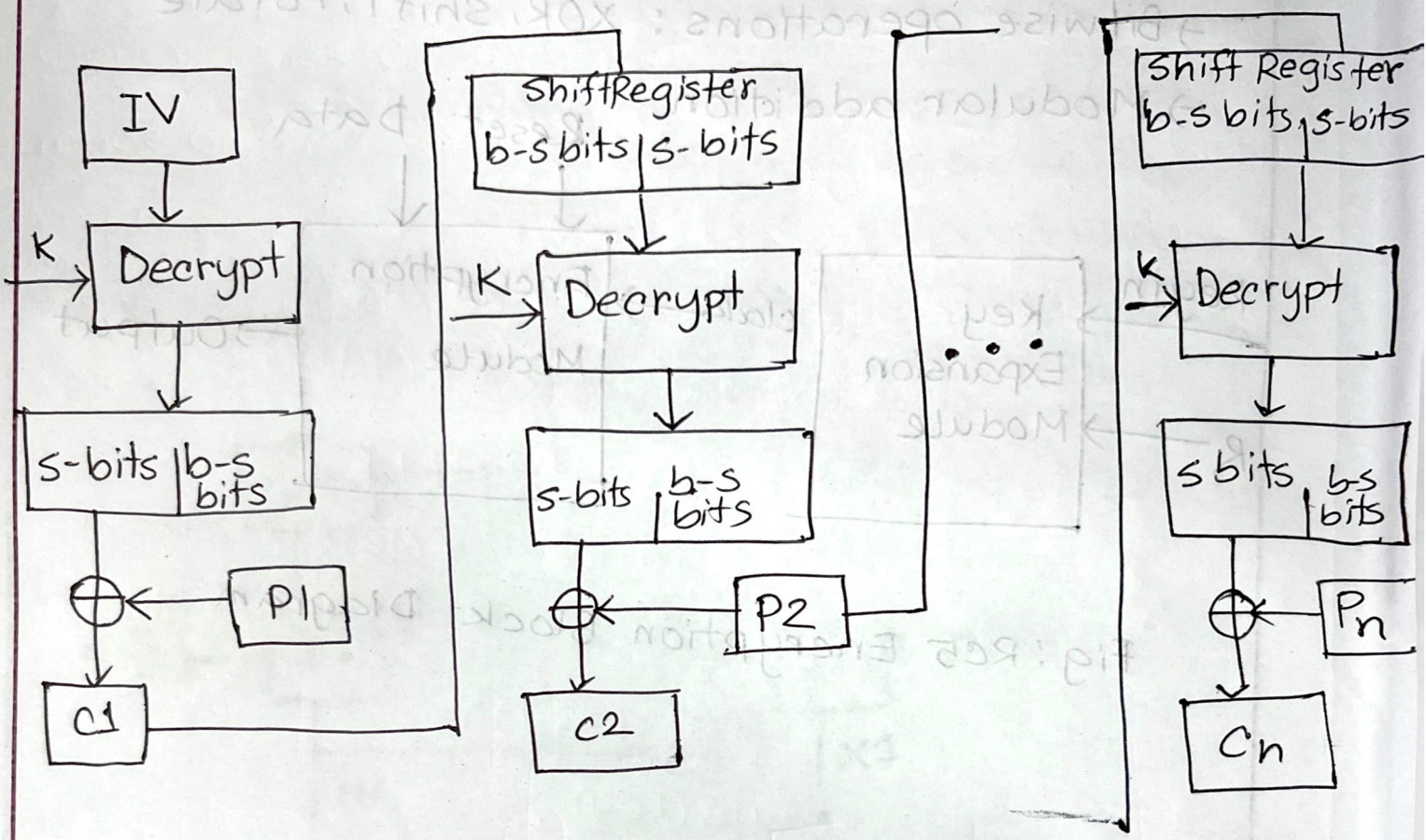
Cipher Feedback Mode (CFM):

In this mode, the cipher is given as feedback to the next block of encryption with some new specifications: first, an initial vector is used for first encryption and output bits are divided as as a set of s and b-s bits.

Diagram:



Decryption



Introduction of RC5 :

RC5 is a fast, simple and secure symmetric key block cipher designed by Ron Rivest in 1994.

Key Features :

- Parameterizable
  - → Word Size (32 bits)
  - → Number of round (12)
  - → Key length (8 bytes)

- Uses
  - → Bitwise operations : XOR, shift, rotate
  - → Modular addition

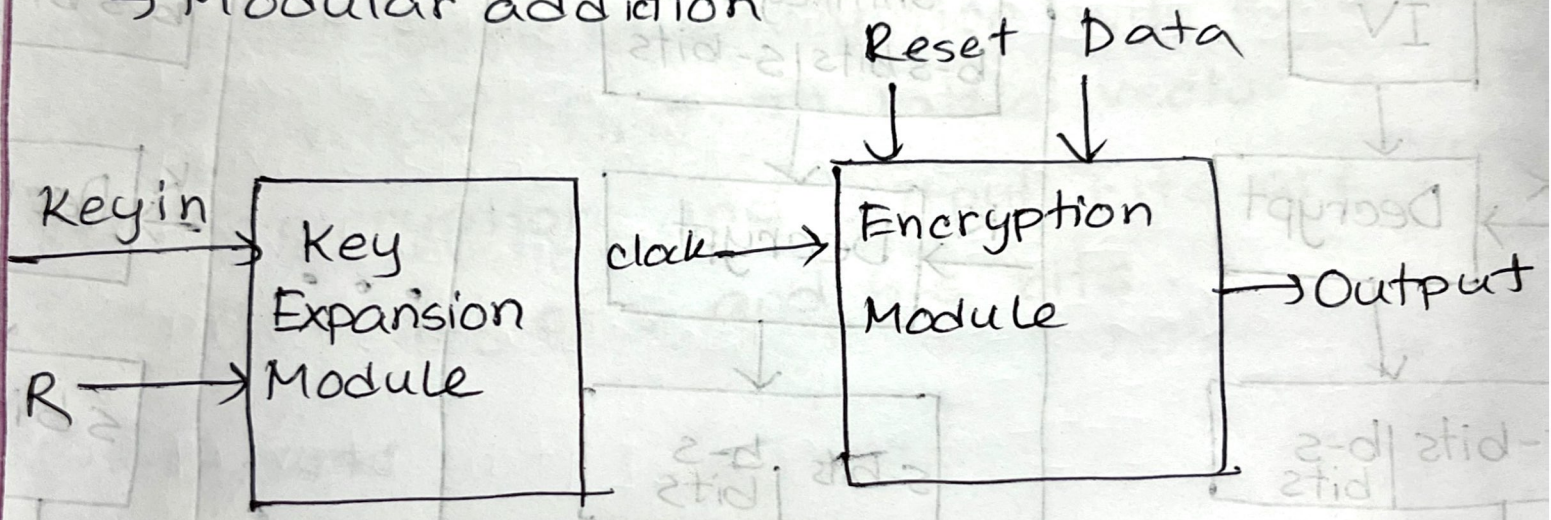
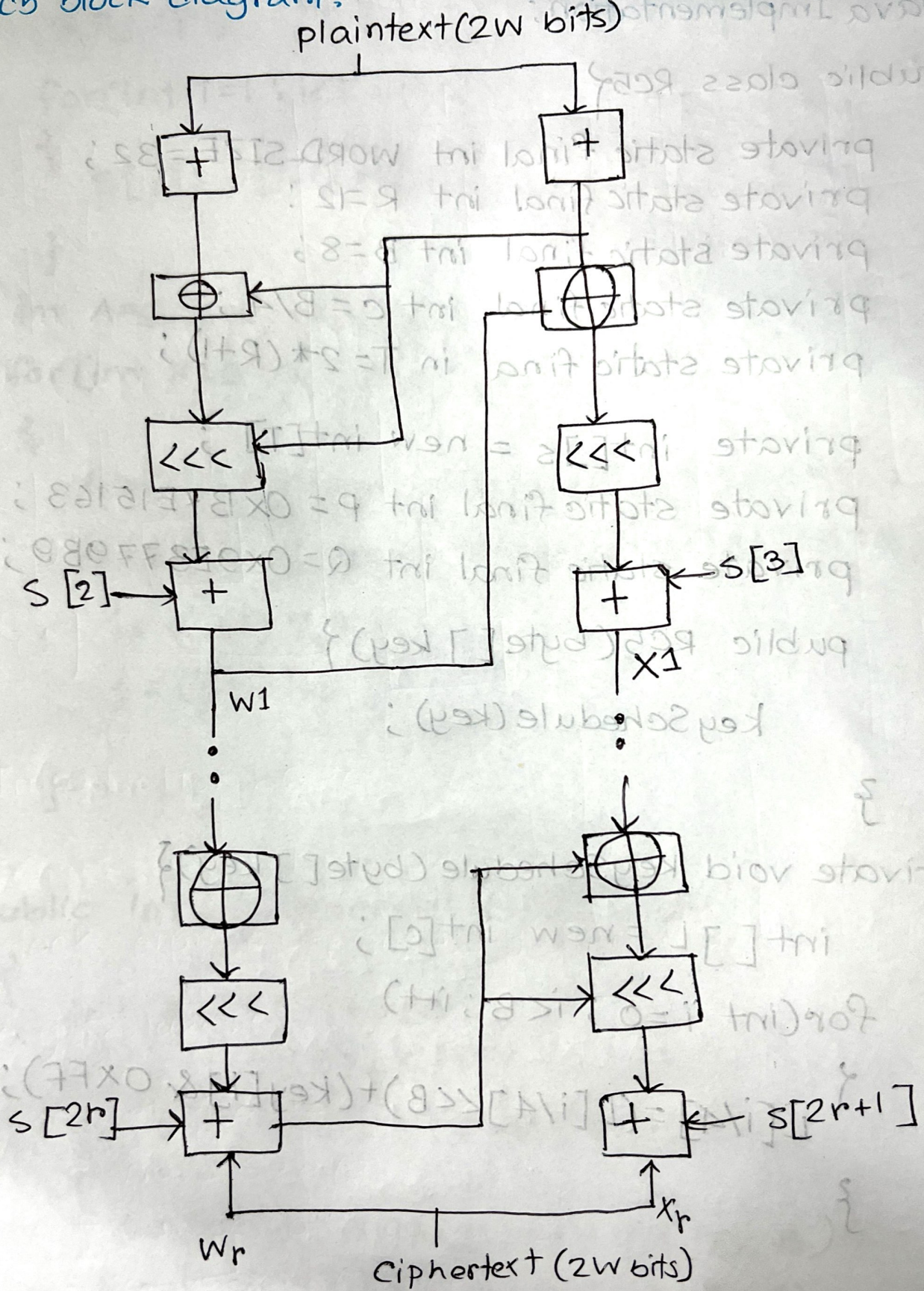


Fig : RC5 Encryption Block Diagram

RC5 block diagram:



Java Implementation:

```
public class RC5{

    private static final int WORD_SIZE =32;
    private static final int R=12 :
    private static final int B=8;
    private static final int c=B/4;
    private static final in T=2*(R+1);

    private int[]s = new int[T];
    private static final int P=0xB7E15163;
    private static final int Q=0x9E3779B9;

    public RC5(byte[] key){
        keySchedule(key);
    }
private void keySchedule(byte[]key){
    int[]L =new int[c];
    for(int i=0; i<B; i++)
    {
        L[i/4]=(L[i/4]<<B)+(key[i]& 0xFF);
    }
```

```
S[0] = P;
for(int i=1; i<T; i++)
{
    S[i] = S[i-1] + Q;
}
int A=0, B=0, i=0, j=0;
for(int k=0; k<3*T; k++)
{
    A = S[i] = Integer.rotateleft((S[i]+A+B), 3);
    B = L[j] = Integer.rotateleft((L[j]+A+B), (A+B));
    i = (i+1) % T;
    j = (j+1) % C;
}
}
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];
    }
    return new int[]{A, B};
}
```

```
public int[] decrypt (int[] ct)
{
    int B=ct[1];
    int A=ct[0];
    for(int i=R; i>=1; i--)
    {
        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="password".getBytes();
    RC5 rc5=new RC5(key);
    int[] pt={0x12345978,0x9abcdef0};
    int[]ct=rc5.encrypt(pt);
System.out.printf("Encrypted: %08x %08x\n",ct[0],
                                    ct[1]);
int[] dt=rc5.decrypt(ct);
```

```
        System.out.println("Decrypted : %08X %08X\n",
                                    dt[0], dt[1]);
    }
}
```

Sample output:

```
Encrypted : 7f93d8c2  1423ba29
Decrypted : 12345678  9abcdef0
```