Cryptography and Cyber Law

Assignment

Shusmita Ghosh Shoili

IT-21006

** Modes of operation and RC5 block-diagram and java implementation and output.

⟹ Modes of operation:

The block ciphers take a fixed size input block and produces a fixed size output block using a transformation that depends on a key. Modes of operation are used to securely process large data by using a

block cipher repeatedly.

Common modes:

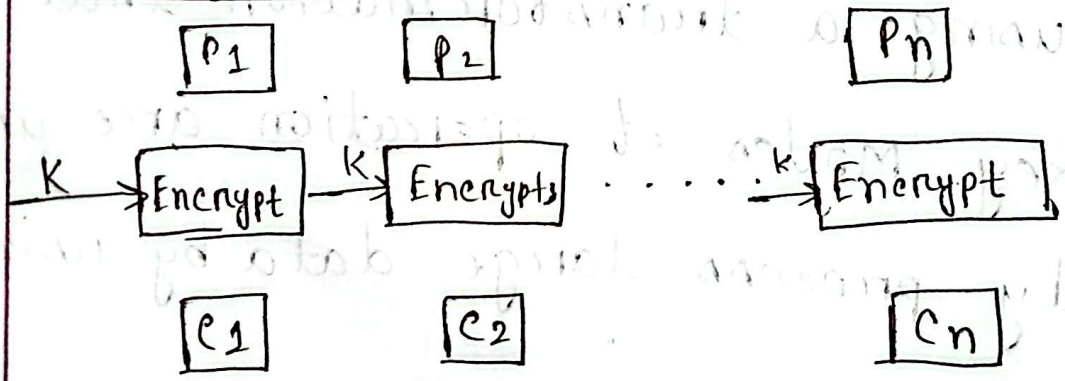
(i) ECB
(ii) CBC
(iii) CFB
(iv) OFB
(v) CTR

Description:-

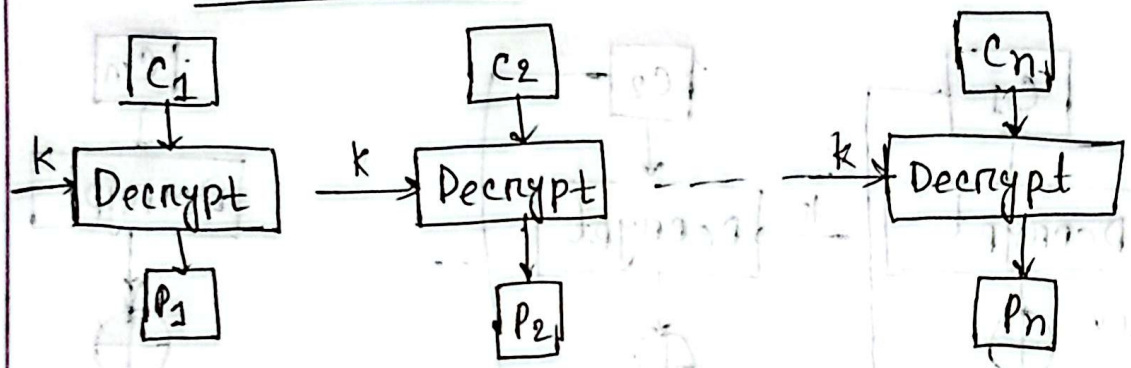
(i) **ECB:** Each block is encrypted independently. Not secure for patterns.

Procedure:

Encryption:



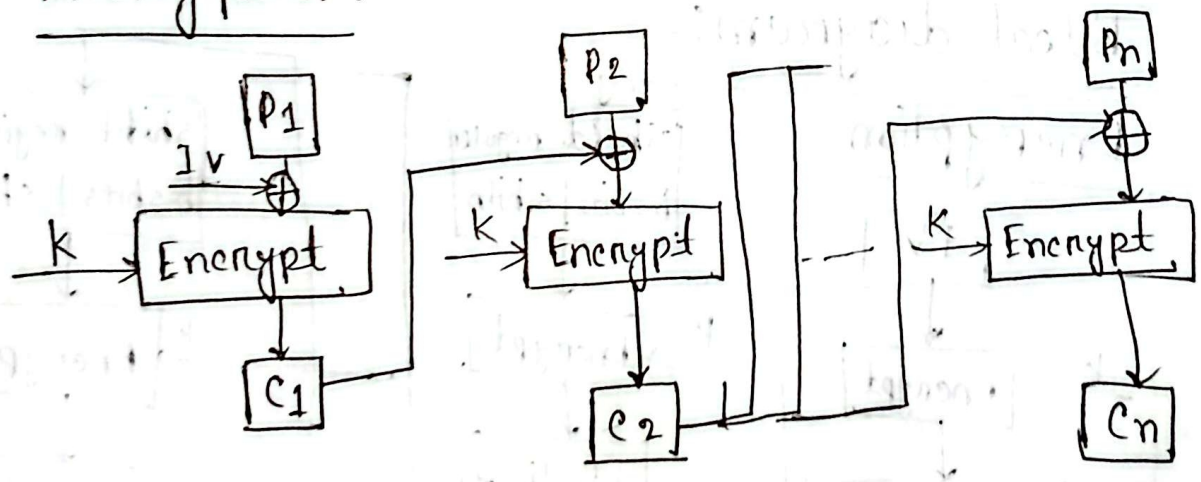
Decryption:



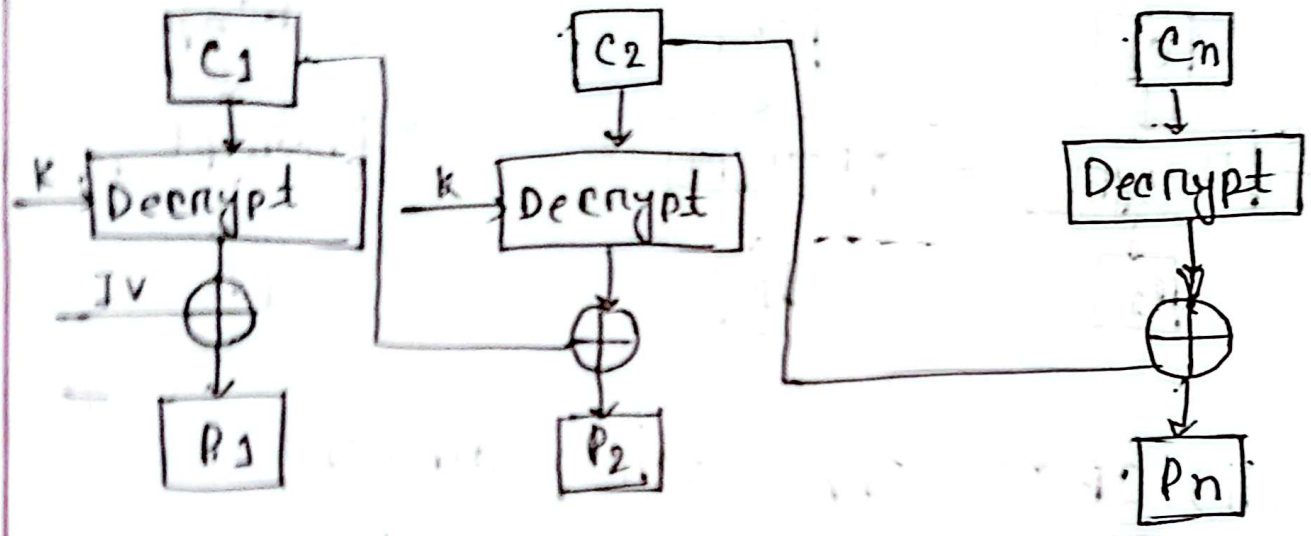
(ii) CBC: XORs each plaintext block with previous ciphertext block before encryption.

Procedure:

Encryption:



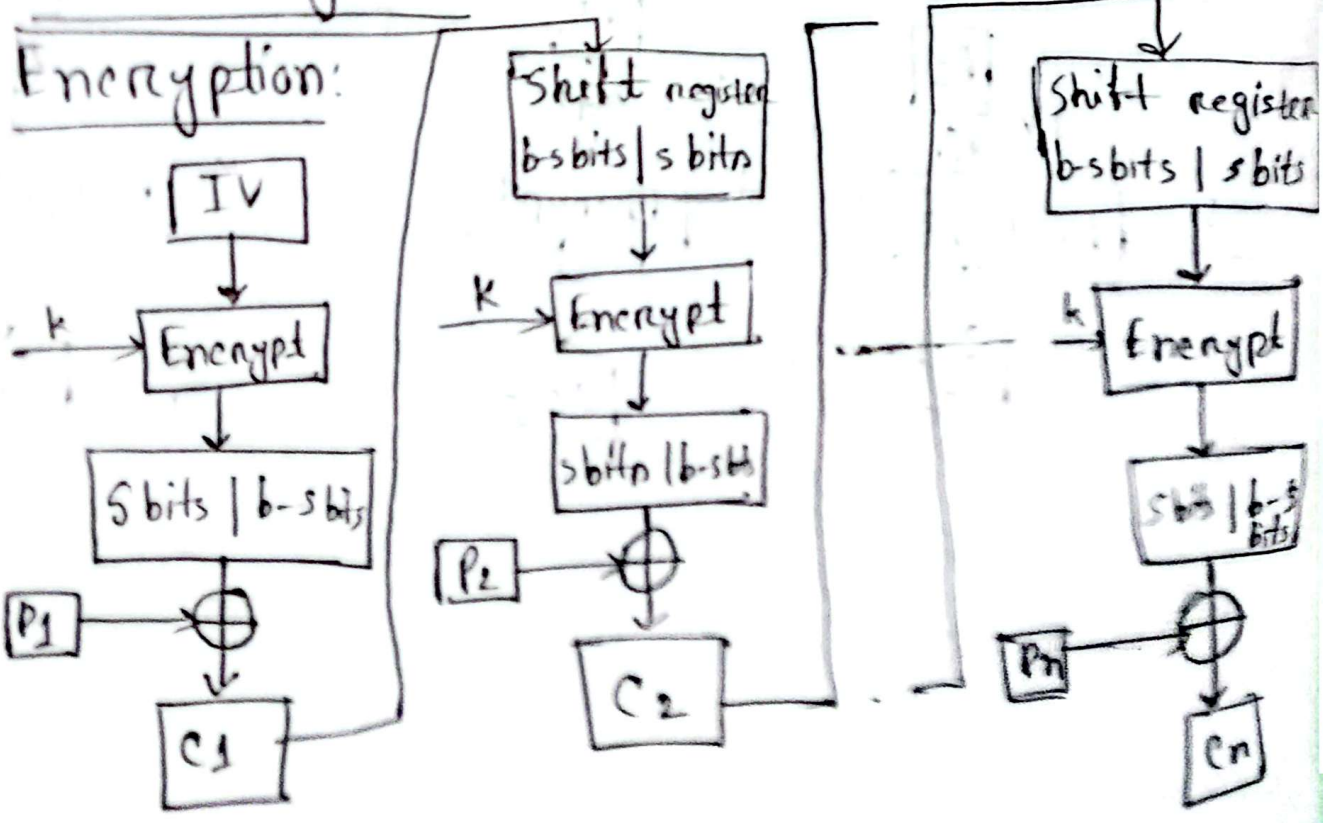
Decryption



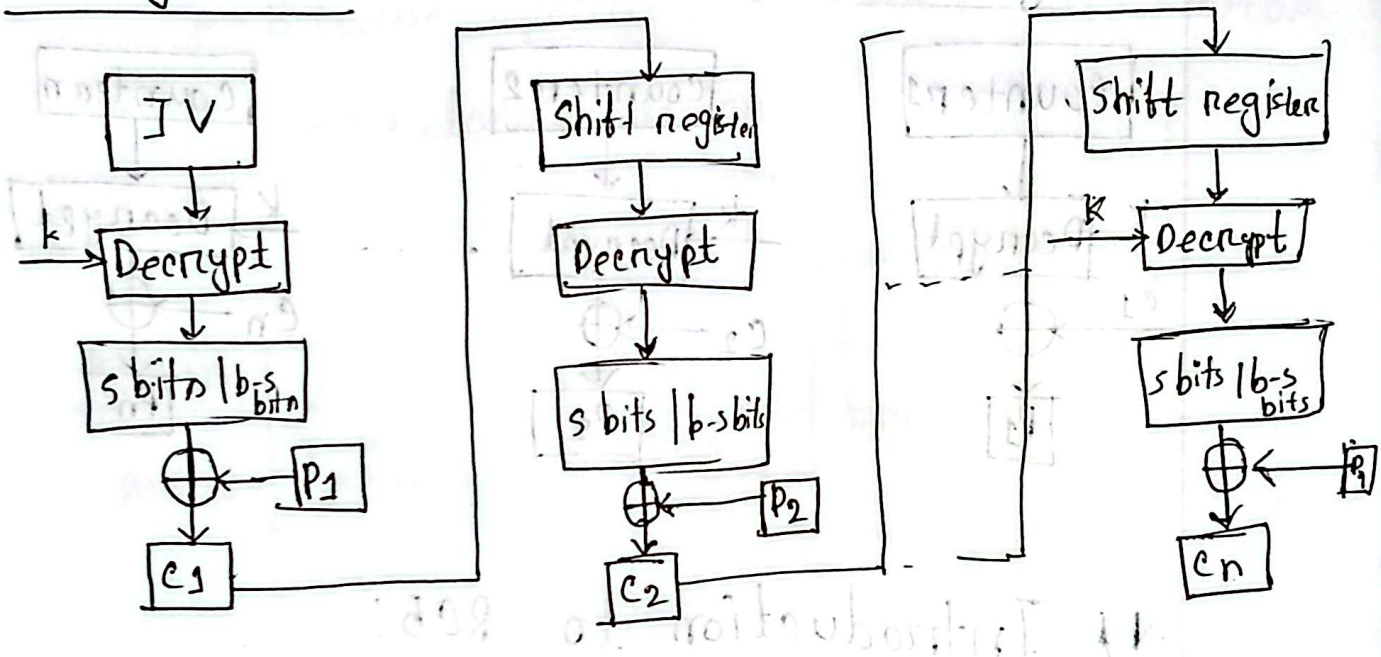
(iii) CFB: Converts block cipher into a self synchronized stream cipher.

Block diagram:-

Encryption:



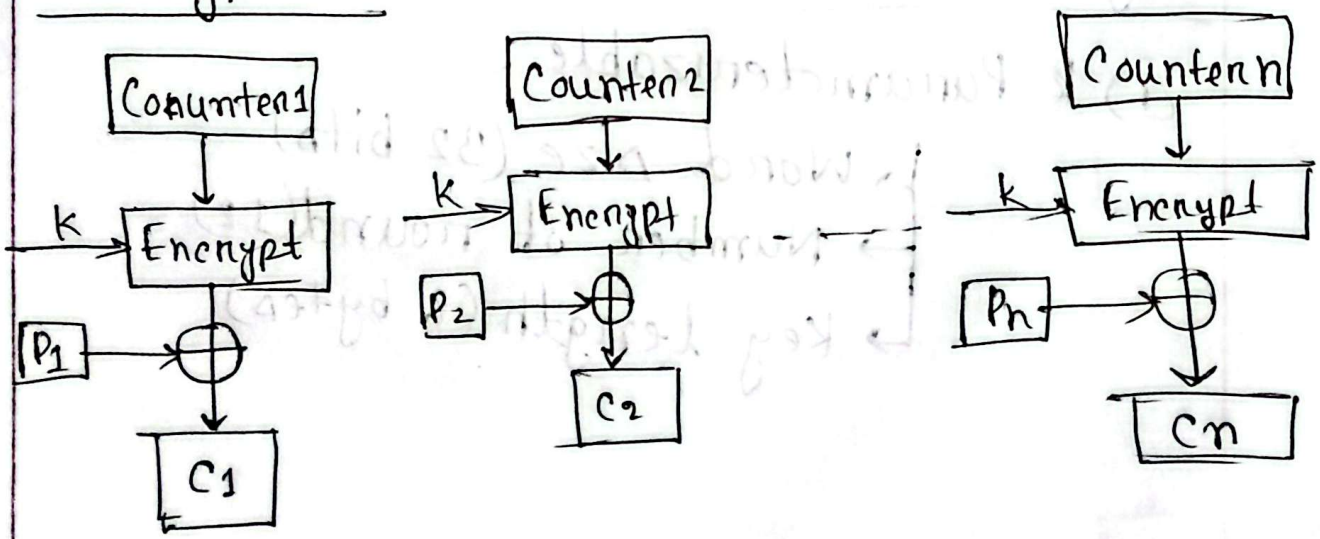
Decryption:



(V) CTR: Uses a counter that gets encrypted and X-ORed with plaintext. Fast and parallelizable.

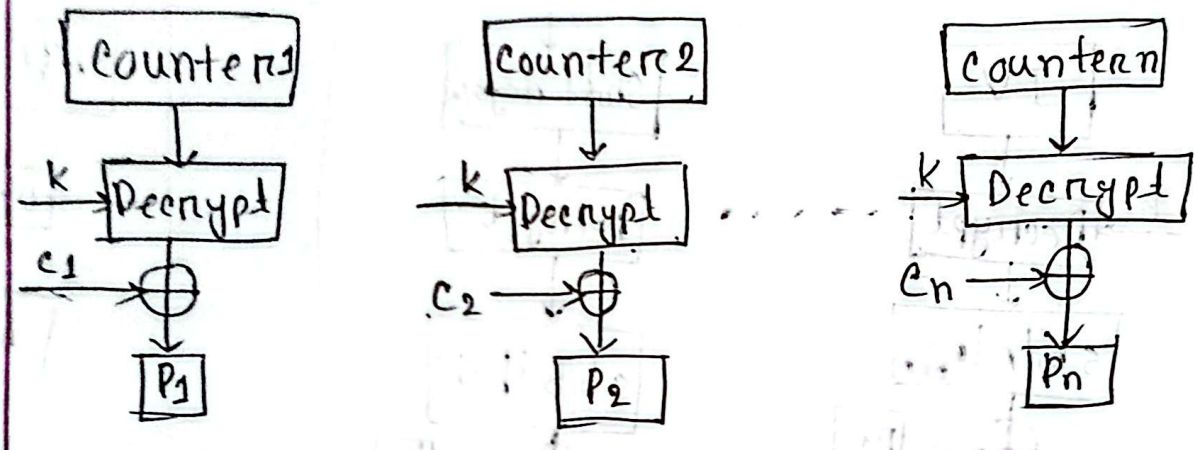
Block diagram:

Encryption:



IT-21006

Decryption:



** Introduction to RC5:

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

Key features:

(i) Parameterizable:

- Word size (32 bits)
- Number of round (12)
- Key length (8 bytes)

(ii) Uses:

- Bitwise operations: X-OR, shift, rotate
- Modular addition

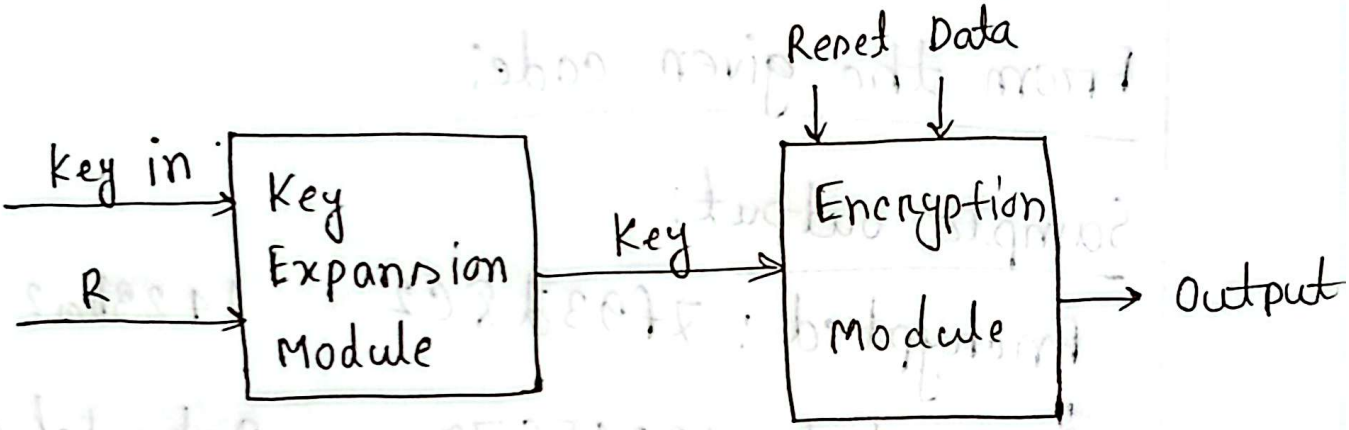
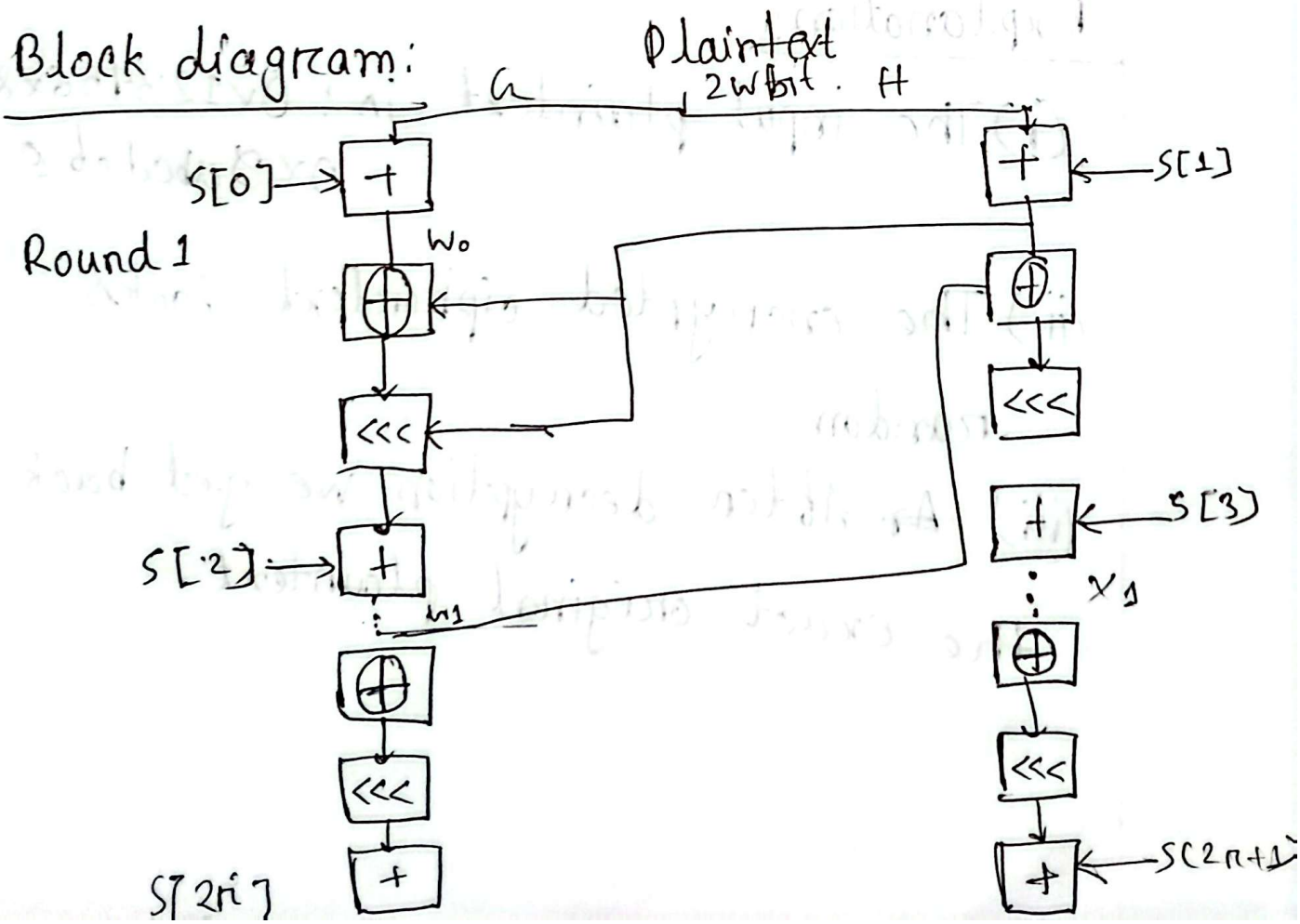


Fig: RC5 Encryption ~~Block diagram~~

Block diagram:



Java Implementation

```
Public class RC5 {
    private static final int WORDSIZE = 32;
    private static final int R = 12;
    private static final int B = 8;
    private static final int C = B/4;
    private static final int 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]<<8)+(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 = {0x12345678, 0x9abcdef0};
int[] ct = rc5.encrypt(pt);
```

```
System.out.printf("Encrypted : %08x
        %08x\n", ct[0], ct[1]);

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

Sample Output :

```
Encrypted : 7b93d8c2 1423ba29
Decrypted : 12345678 9abcdef0
```

Explanation :-

- The input plaintext is : 0x12345678 0x9abcdef0
- The encrypted ciphertext looks random.
- After decryption, we get back the exact original plaintext.