Modes of Operation

CS6025 Data Encoding

Yizong Cheng

2-26-15

Modes of Operation

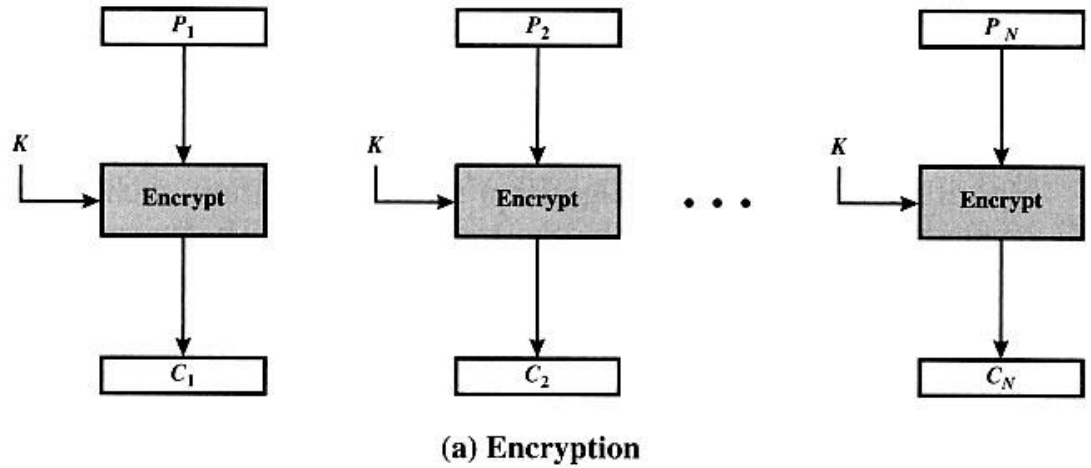
A mode of operation is a technique for enhancing the effect of a cryptographic algorithm or adapting the algorithm for an application, such as applying a block cipher to a sequence of data blocks or a data stream.

Five modes of operation have been standardized by NIST for use with symmetric block ciphers such as DES and AES: electronic codebook mode, cipher block chaining mode, cipher feedback mode, output feedback mode, and counter mode.

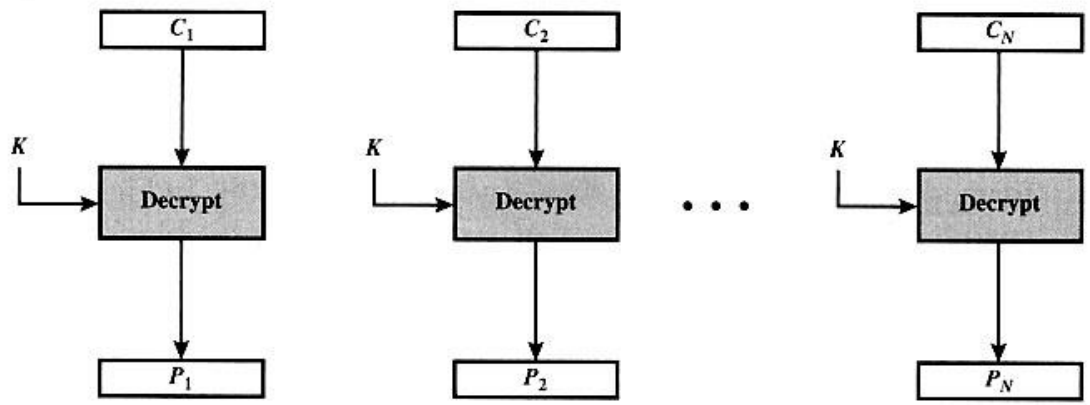
Another important mode, XTS-AES, has been standardized by the IEEE Security in Storage Working Group (P1619). The standard describes a method of encryption for data stored in sector-based devices where the threat model includes possible access to stored data by the adversary.

Electronic Codebook (ECB)	Each block of 64 plaintext bits is encoded independently using the same key.	• Secure transmission of single values (e.g., an encryption key)
Cipher Block Chaining (CBC)	The input to the encryption algorithm is the XOR of the next 64 bits of plaintext and the preceding 64 bits of ciphertext.	• General-purpose block-oriented transmission • Authentication
Cipher Feedback (CFB)	Input is processed s bits at a time. Preceding ciphertext is used as input to the encryption algorithm to produce pseudorandom output, which is XORed with plaintext to produce next unit of ciphertext.	• General-purpose stream-oriented transmission • Authentication
Output Feedback (OFB)	Similar to CFB, except that the input to the encryption algorithm is the preceding encryption output, and full blocks are used.	• Stream-oriented transmission over noisy channel (e.g., satellite communication)
Counter (CTR)	Each block of plaintext is XORed with an encrypted counter. The counter is incremented for each subsequent block.	• General-purpose block-oriented transmission • Useful for high-speed requirements

Electronic Codebook (ECB) Mode



(a) Encryption



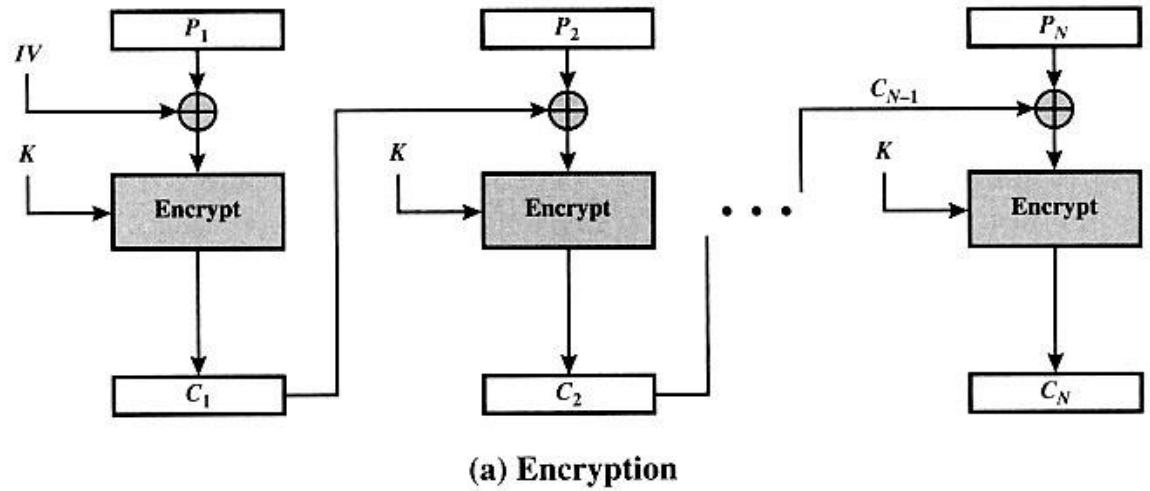
(b) Decryption

Figure 6.3 Electronic Codebook (ECB) Mode

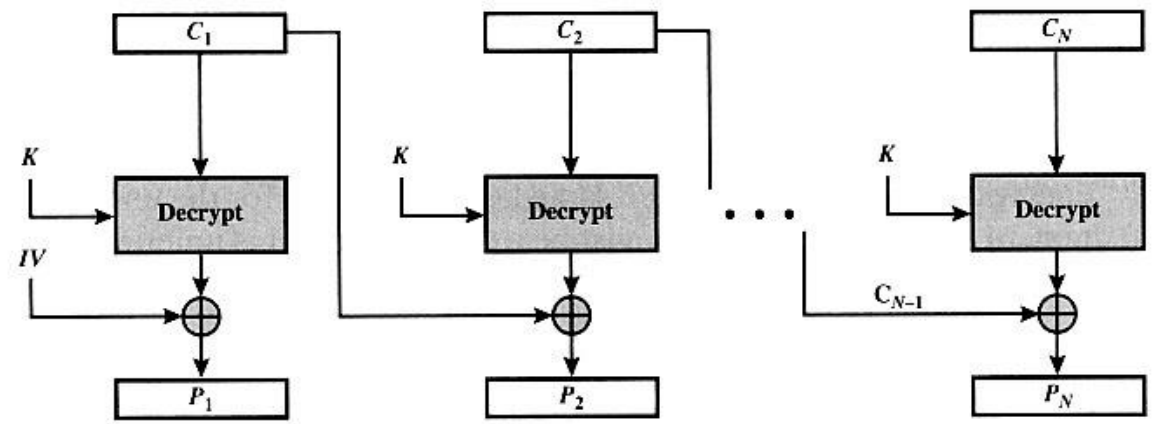
Electronic Codebook (ECB)

ECB	$C_j = E(K, P_j)$ $j = 1, \ldots, N$	$P_j = D(K, C_j)$ $j = 1, \ldots, N$

Cipher Block Chaining (CBC)



(a) Encryption



(b) Decryption

Figure 6.4 Cipher Block Chaining (CFB) Mode

Cipher Block Chaining (CBC)

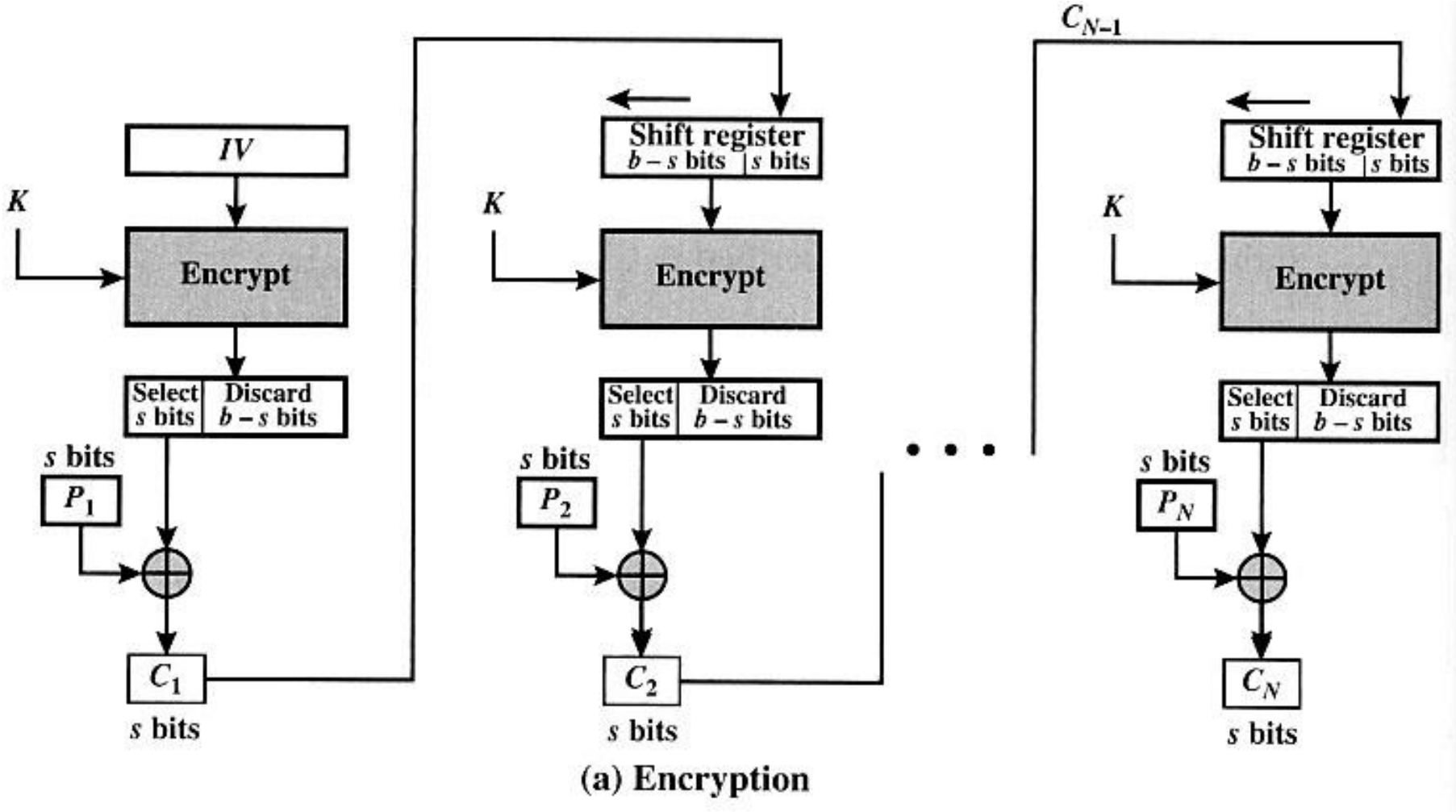
CBC	$C_1 = \mathrm{E}(K, [P_1 \oplus \mathrm{IV}])$ $C_j = \mathrm{E}(K, [P_j \oplus C_{j-1}])\ j = 2, \ldots, N$	$P_1 = \mathrm{D}(K, C_1) \oplus \mathrm{IV}$ $P_j = \mathrm{D}(K, C_j) \oplus C_{j-1}\ \ j = 2, \ldots, N$

```
void addBlock(int[] destination, int[] source){
  for (int k = 0; k < blockSize; k++)
     destination[k] ^= source[k];
}

void copyBlock(int[] destination, int[] source){
  for (int k = 0; k < blockSize; k++)
     destination[k] = source[k];
}


 void encrypt(){
   int[] lastBlock = new int[blockSize];
   for (int k = 0; k < blockSize; k++) lastBlock[k] = 0;
   while (readBlock() > 0){
     addBlock(state, lastBlock);
     blockCipher();
     writeBlock();
     copyBlock(lastBlock, state);
   }
   System.out.flush();
 }
```

Cipher Feedback (CFB) Mode



(a) Encryption

Cipher Feedback (CFB) Decryption

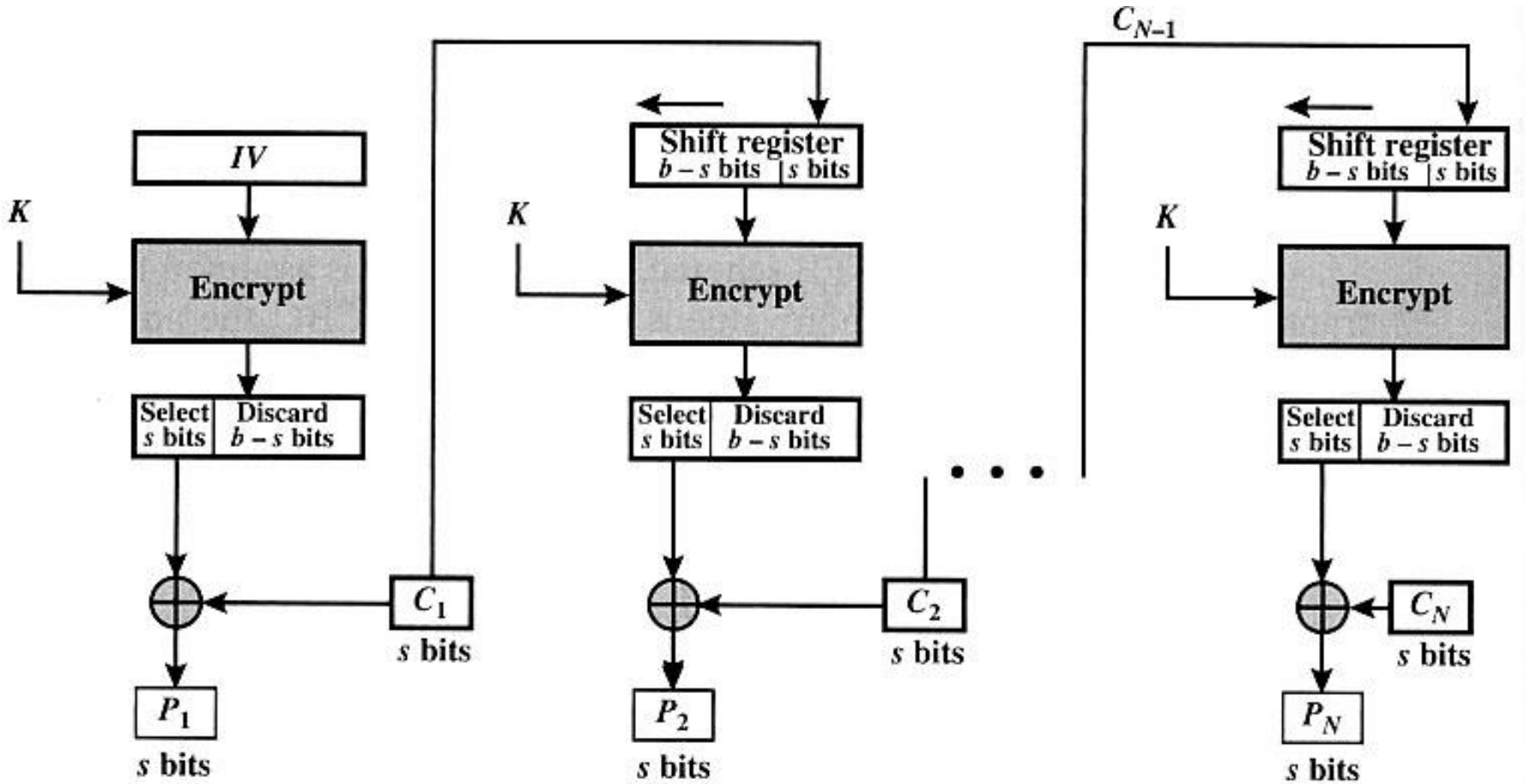


Figure 6.5 *s*-bit Cipher Feedback (CFB) Mode

Cipher Feedback (CFB) Mode

CFB	$I_1 = IV$ $I_j = \text{LSB}_{b-s}(I_{j-1}) \parallel C_{j-1}$ $j = 2, \ldots, N$ $O_j = \text{E}(K, I_j)$ $j = 1, \ldots, N$ $C_j = P_j \oplus \text{MSB}_s(O_j)$ $j = 1, \ldots, N$	$I_1 = IV$ $I_j = \text{LSB}_{b-s}(I_{j-1}) \parallel C_{j-1}$ $j = 2, \ldots, N$ $O_j = \text{E}(K, I_j)$ $j = 1, \ldots, N$ $P_j = C_j \oplus \text{MSB}_s(O_j)$ $j = 1, \ldots, N$

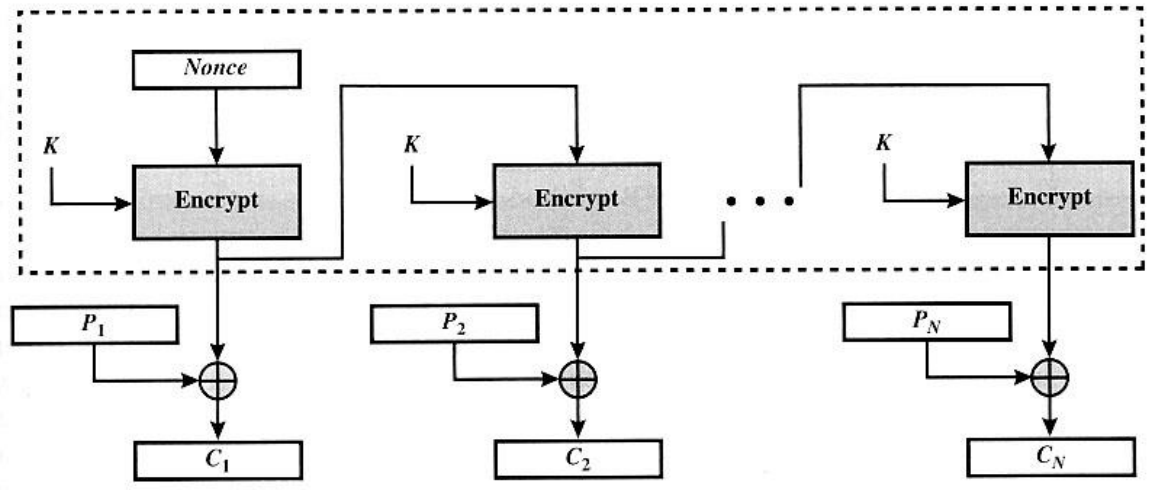
```
int readBlock(){
  byte[] data = new byte[blockSize];
  int len = 0;
  try {
    len = System.in.read(data);
  } catch (IOException e){
    System.err.println(e.getMessage());
    System.exit(1);
  }
  if (len <= 0) return len;
  for (int i = 0; i < len; i++){
    if (data[i] < 0) inBlock[i] = data[i] + fieldSize;
    else inBlock[i] = data[i];
  }
  return len;
}
```

```
void encrypt(){
   for (int k = 0; k < blockSize; k++) register[k] = 0; // iv = 0
   int len = 0;
   while ((len = readBlock()) >= 0){
     for (int i = 0; i < len; i++){
       copyBlock(state, register);
       blockCipher();
       outBlock[i] = inBlock[i] ^ state[0];
       shiftRegister(outBlock[i]);
     }
     writeBlock(len);
   }
 }
```

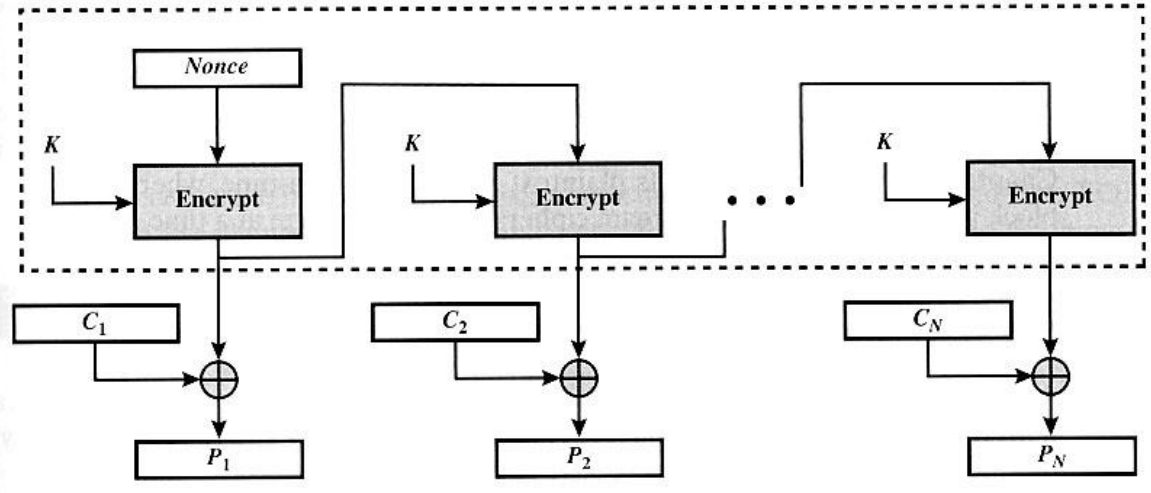
```
void writeBlock(int len){
  byte[] data = new byte[blockSize];
  for (int i = 0; i < len; i++)
    data[i] = (byte)(outBlock[i]);
  System.out.write(data, 0, len);
}

void shiftRegister(int newValue){
  for (int i = 1; i < blockSize; i++)
    register[i - 1] = register[i];
  register[blockSize - 1] = newValue;
}
```

Output Feedback (OFB) Mode



(a) Encryption



(b) Decryption

Figure 6.6 Output Feedback (OFB) Mode

Output Feedback (OFB) Mode

OFB	$I_1 = Nonce$ $I_j = O_{j-1}$ $j = 2, \ldots, N$ $O_j = \mathrm{E}(K, I_j)$ $j = 1, \ldots, N$ $C_j = P_j \oplus O_j$ $j = 1, \ldots, N-1$ $C_N^* = P_N^* \oplus \mathrm{MSB}_u(O_N)$	$I_1 = Nonce$ $I_j = \mathrm{LSB}_{b-s}(I_{j-1}) \| C_{j-1}$ $j = 2, \ldots, N$ $O_j = \mathrm{E}(K, I_j)$ $j = 1, \ldots, N$ $P_j = C_j \oplus O_j$ $j = 1, \ldots, N-1$ $P_N^* = C_N^* \oplus \mathrm{MSB}_u(O_N)$

```
void readNonce(String filename){
   Scanner in = null;
   try {
     in = new Scanner(new File(filename));
   } catch (FileNotFoundException e){
     System.err.println(filename + " not found");
     System.exit(1);
   }
   hexkey = in.nextLine();
   in.close();
   for (int i = 0; i < blockSize; i++) state[i] =
     Integer.parseInt(hexkey.substring(i * 2, (i + 1) * 2), 16);
 }
```

```
void encrypt(){
  int len = 0;
  while ((len = readBlock()) >= 0){
    blockCipher();
    for (int i = 0; i < len; i++)
      outBlock[i] = inBlock[i] ^ state[i];
    writeBlock(len);
  }
}
```

Counter (CTR) Mode

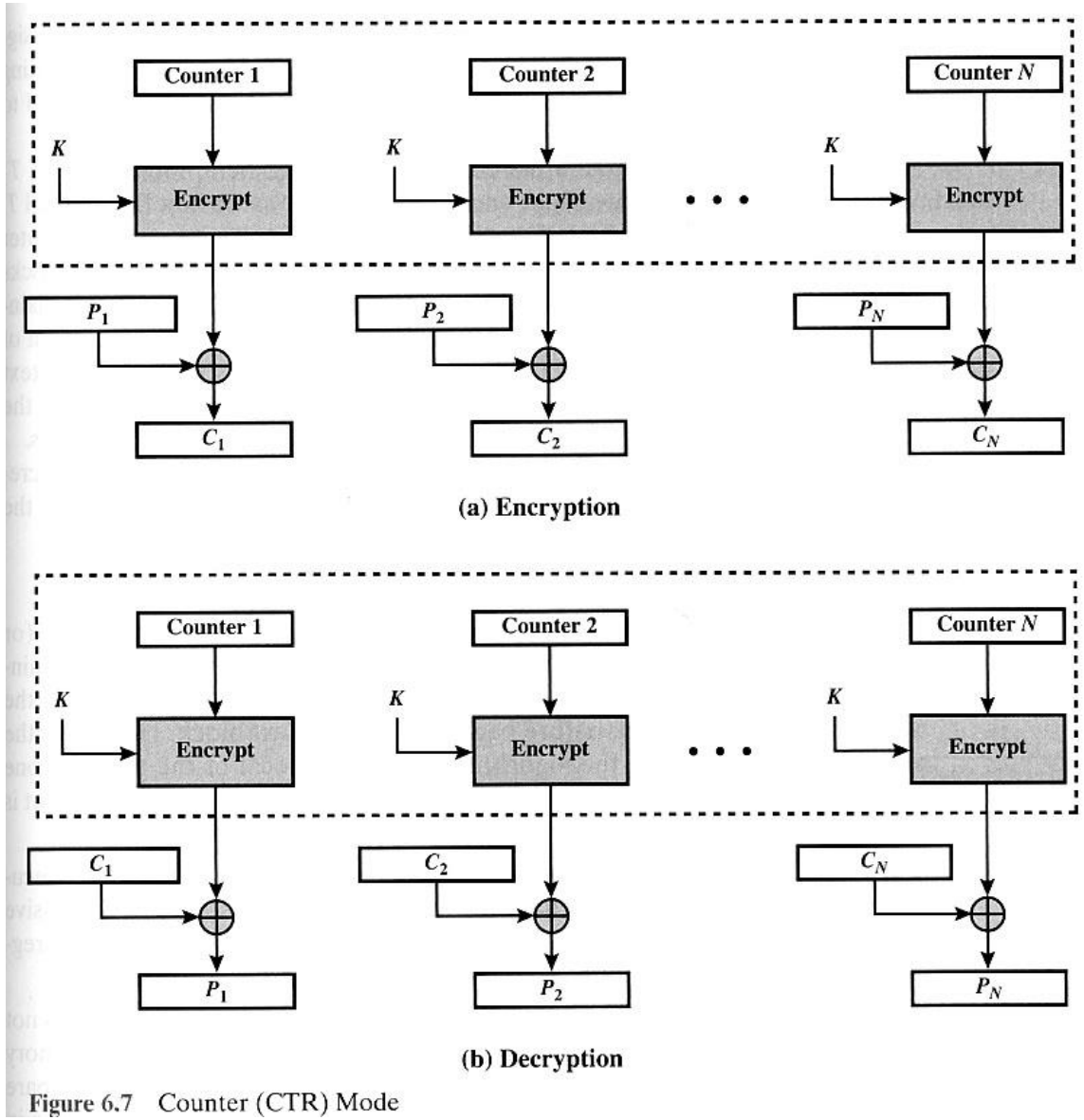


Figure 6.7 Counter (CTR) Mode

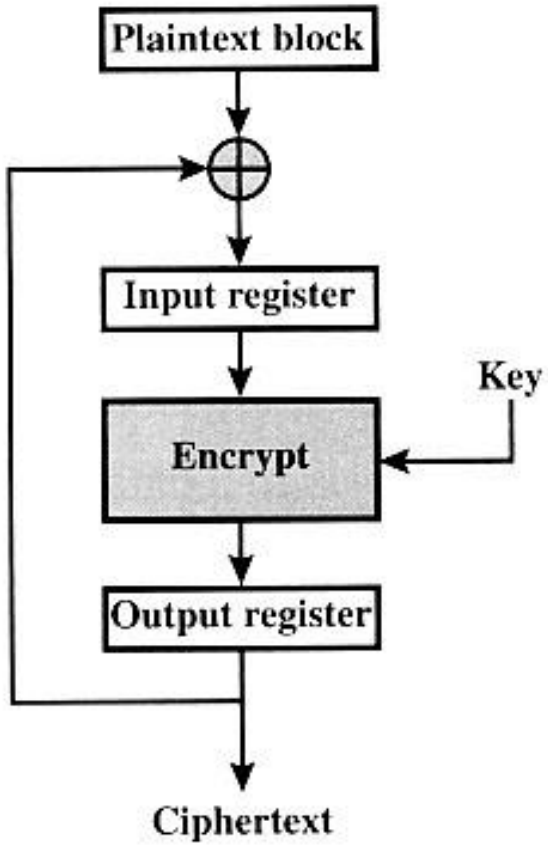
Counter (CTR) Mode

CTR	$C_j = P_j \oplus \mathrm{E}(K, T_j) \quad j = 1, \ldots, N-1$ $C_N^* = P_N^* \oplus \mathrm{MSB}_u[\mathrm{E}(K, T_N)]$	$P_j = C_j \oplus \mathrm{E}(K, T_j) \quad j = 1, \ldots, N-1$ $P_N^* = C_N^* \oplus \mathrm{MSB}_u[\mathrm{E}(K, T_N)]$

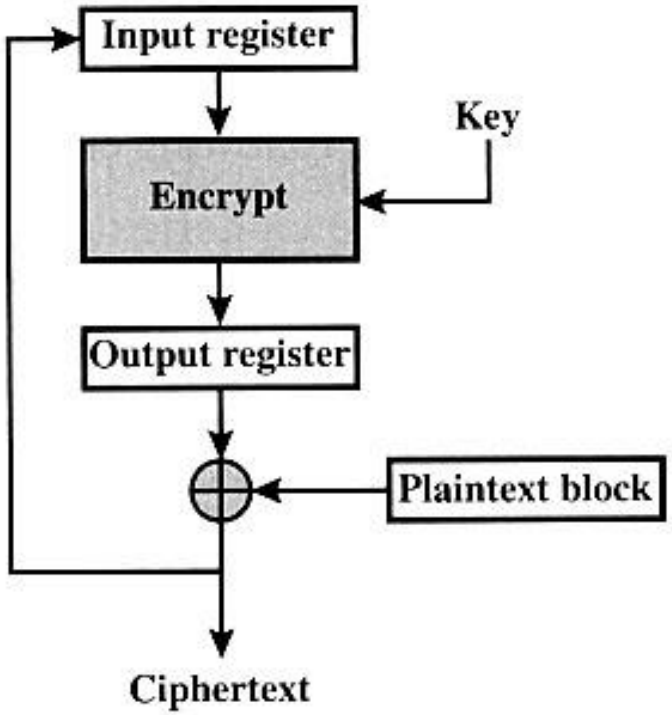
```
void encrypt(){
   int len = 0;
   while ((len = readBlock()) >= 0){
     copyBlock(state, counter);
     blockCipher();
     addBlock(state, inBlock);
     writeBlock(len);
     incrementCounter();
   }
 }
```

```
void incrementCounter(){
   boolean carry = true;
   int k = blockSize - 1;
   while (carry && k >= 0){
     counter[k]++;
     if (counter[k] >= fieldSize) counter[k] = 0;
     else carry = false;
     k--;
   }
   if (carry && k < 0) counter[blockSize - 1] = 1;
 }
```

Feedback Characteristics

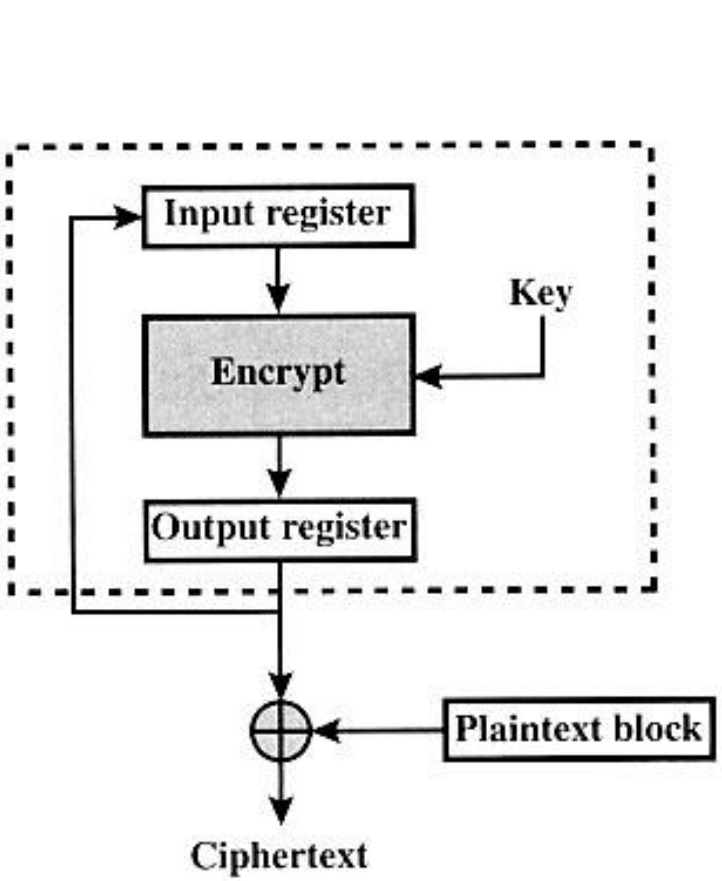


(a) Cipher block chaining (CBC) mode

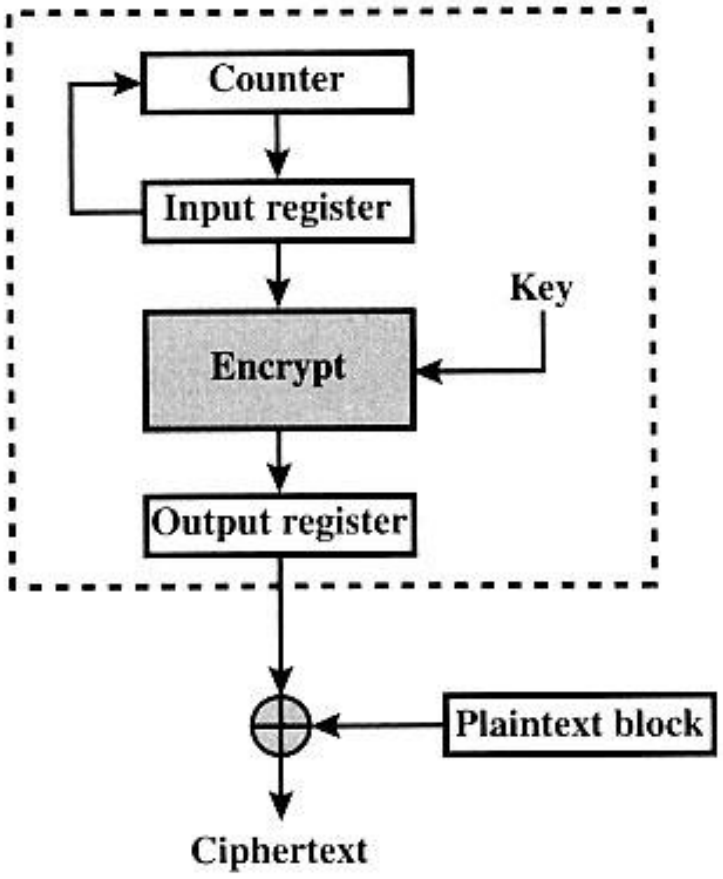


(b) Cipher feedback (CFB) mode

Feedback Characteristics



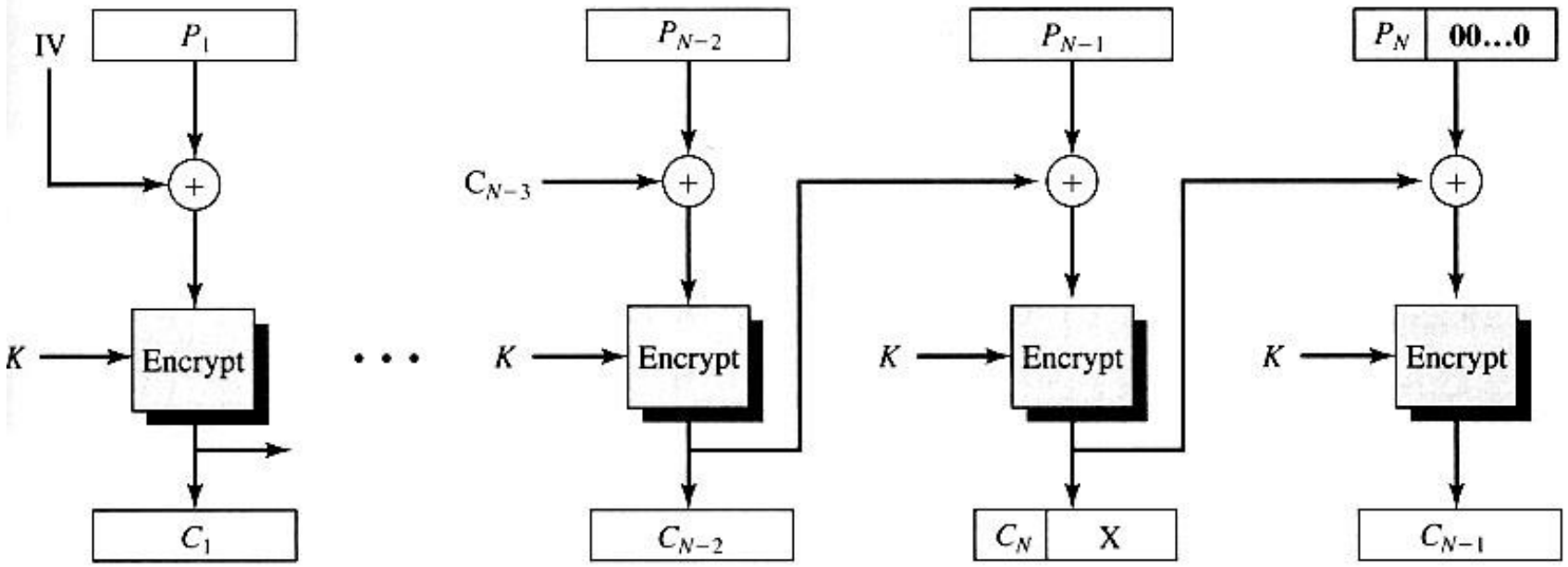
(c) Output feedback (OFB) mode



(d) Counter (CTR) mode

Figure 6.8 Feedback Characteristic of Modes of Operation

Ciphertext Stealing Mode (CTS)

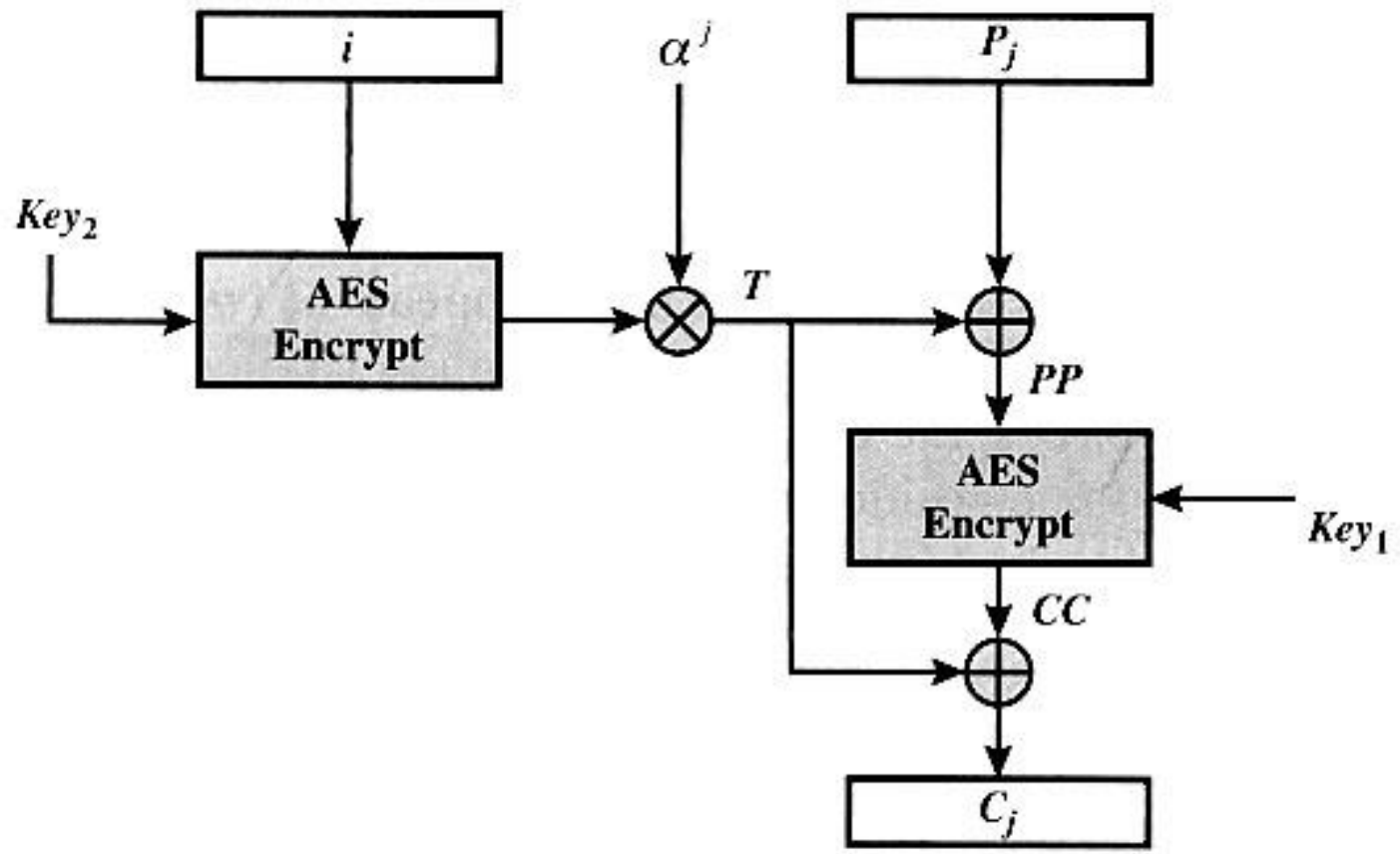


(a) Ciphertext stealing mode

```
void encrypt(){
   int[] lastBlock = new int[blockSize];
   int len = readBlock();
   blockCipher();
   copyBlock(lastBlock, state);
   while ((len = readBlock()) >= 0){
     addBlock(state, lastBlock);
     blockCipher();
     if (len == blockSize){
       writeBlock(lastBlock, blockSize);
       copyBlock(lastBlock, state);
     }else{
        writeBlock(state, blockSize);
        writeBlock(lastBlock, len);
     }
   }
   System.out.flush();
 }
```

1. The ciphertext is freely available for an attacker. Among the circumstances that lead to this situation:
   a. A group of users has authorized access to a database. Some of the records in the database are encrypted so that only specific users can successfully read/write them. Other users can retrieve an encrypted record but are unable to read it without the key.
   b. An unauthorized user manages to gain access to encrypted records.
   c. A data disk or laptop is stolen, giving the adversary access to the encrypted data.
2. The data layout is not changed on the storage medium and in transit. The encrypted data must be the same size as the plaintext data.
3. Data are accessed in fixed sized blocks, independently from each other. That is, an authorized user may access one or more blocks in any order.
4. Encryption is performed in 16-byte blocks, independently from other blocks (except the last two plaintext blocks of a sector, if its size is not a multiple of 16 bytes).
5. There are no other metadata used, except the location of the data blocks within the whole data set.
6. The same plaintext is encrypted to different ciphertexts at different locations, but always to the same ciphertext when written to the same location again.
7. A standard conformant device can be constructed for decryption of data encrypted by another standard conformant device.

XTS-AES Block Encryption

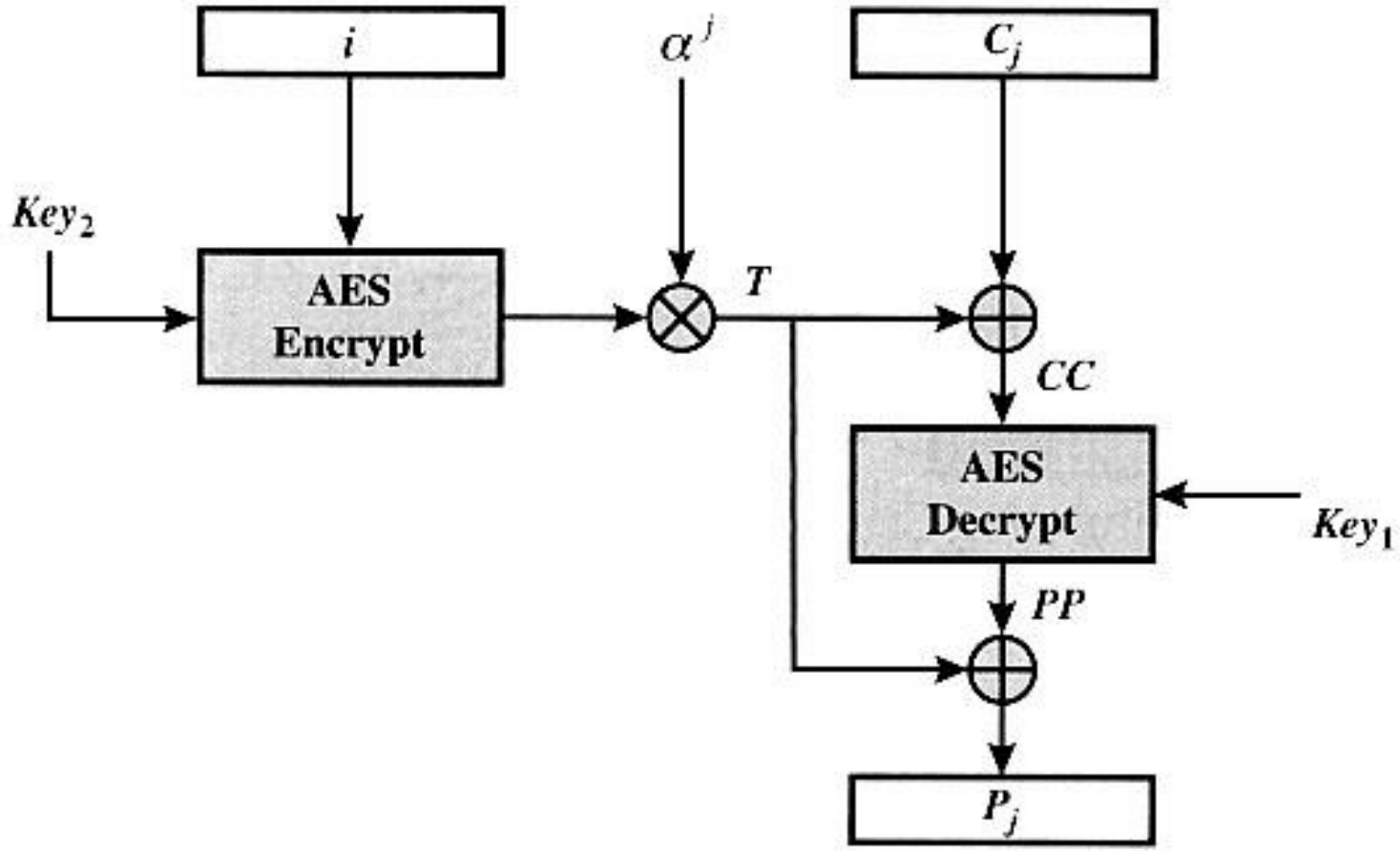


(a) Encryption

Symbols

Key	The 256 or 512 bit XTS-AES key; this is parsed as a concatenation of two fields of equal size called Key_1 and Key_2, such that $Key = Key_1 \| Key_2$.
P_j	The jth block of plaintext. All blocks except possibly the final block have a length of 128 bits. A plaintext data unit, typically a disk sector, consists of a sequence of plaintext blocks $P_1, P_2, \ldots, P_m$.
C_j	The jth block of ciphertext. All blocks except possibly the final block have a length of 128 bits.
j	The sequential number of the 128-bit block inside the data unit.
i	The value of the 128-bit tweak. Each data unit (sector) is assigned a tweak value that is a nonnegative integer. The tweak values are assigned consecutively, starting from an arbitrary nonnegative integer.
α	A primitive element of GF(2^{128}) that corresponds to polynomial x (i.e., $0000 \ldots 010_2$).
α^j	a multiplied by itself j times, in GF(2^{128}).
$\oplus$	Bitwise XOR.
$\otimes$	Modular multiplication of two polynomials with binary coefficients modulo $x^{128} + x^7 + x^2 + x + 1$. Thus, this is multiplication in GF(2^{128}).

XTS-AES Block Decryption



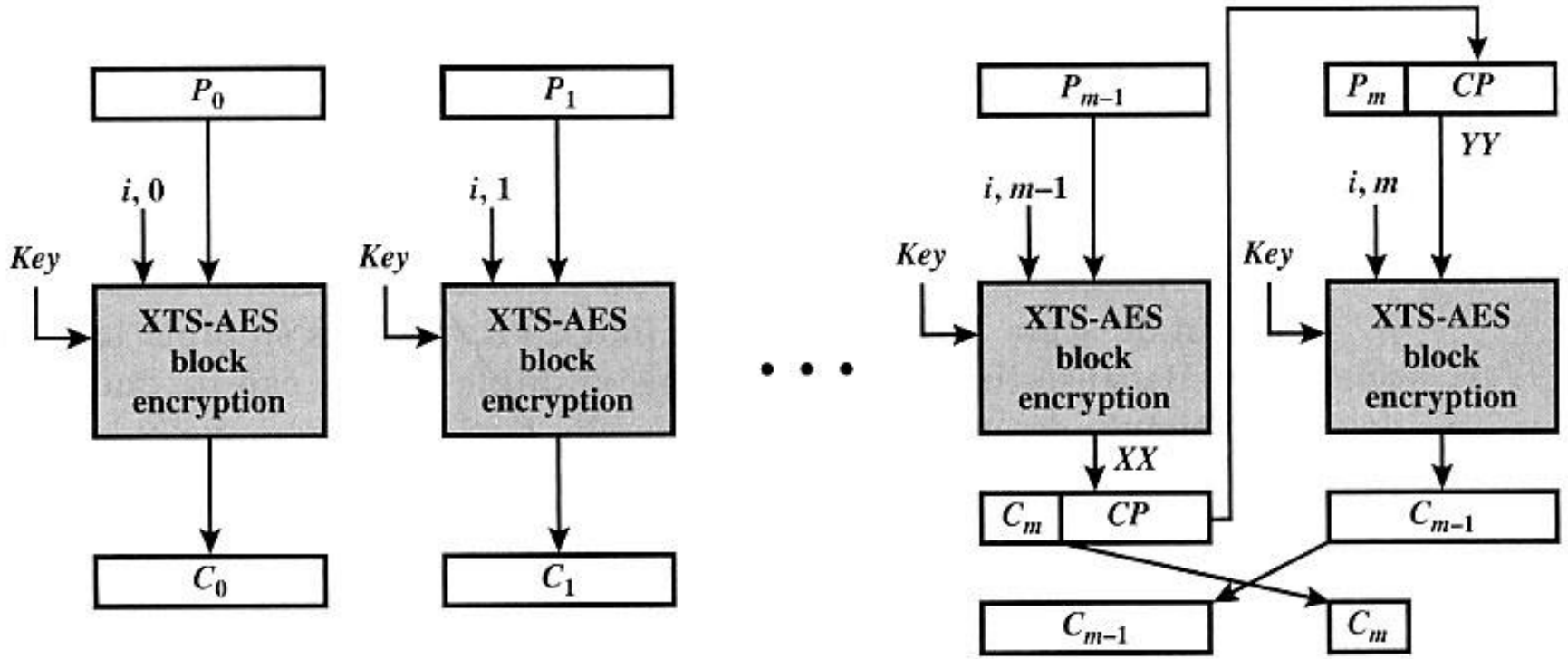
(b) Decryption

Figure 6.9 XTS-AES Operation on Single Block

XTS-AES Block Operation

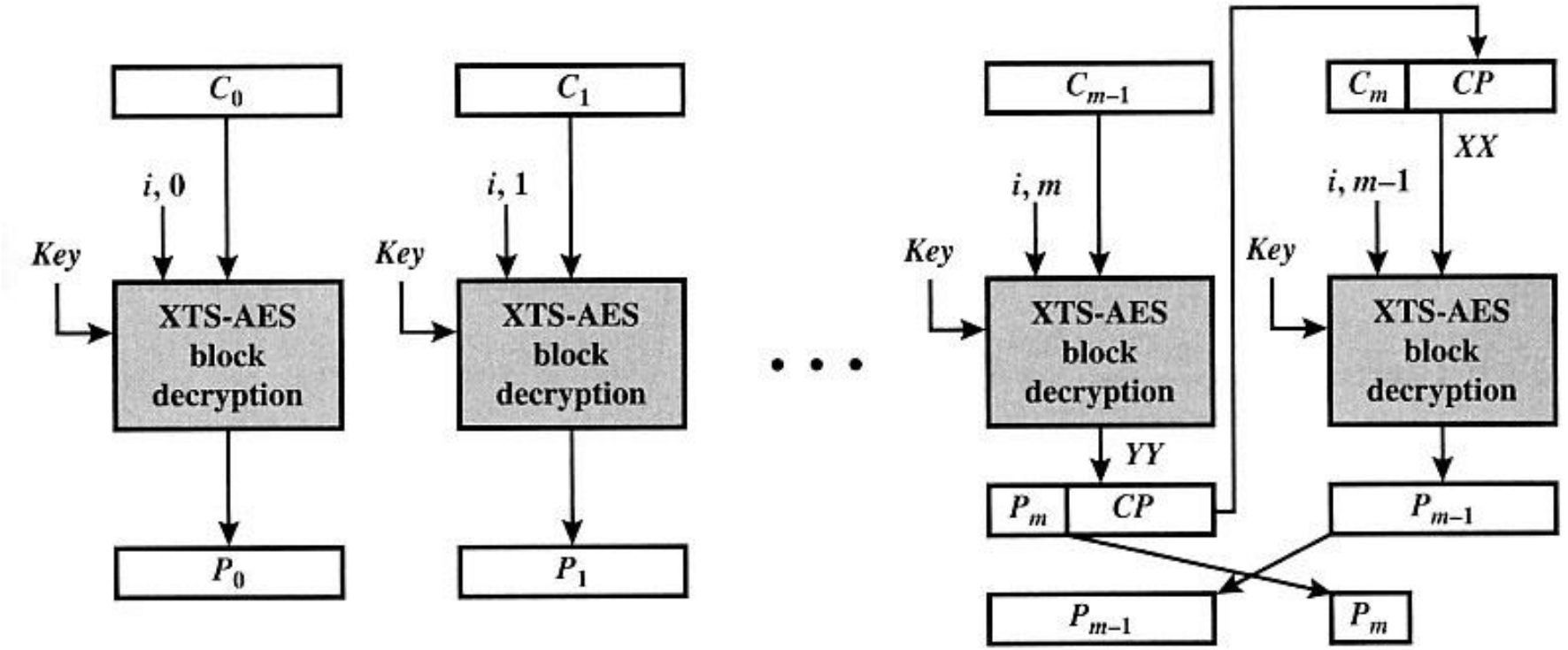
XTS-AES block operation	$T = \mathrm{E}(K_2, i) \otimes \alpha^j$ $PP = P \oplus T$ $CC = \mathrm{E}(K_1, PP)$ $C = CC \oplus T$	$T = \mathrm{E}(K_2, i) \otimes \alpha^j$ $CC = C \oplus T$ $PP = \mathrm{D}(K_1, CC)$ $P = PP \oplus T$

XTS-AES Encryption



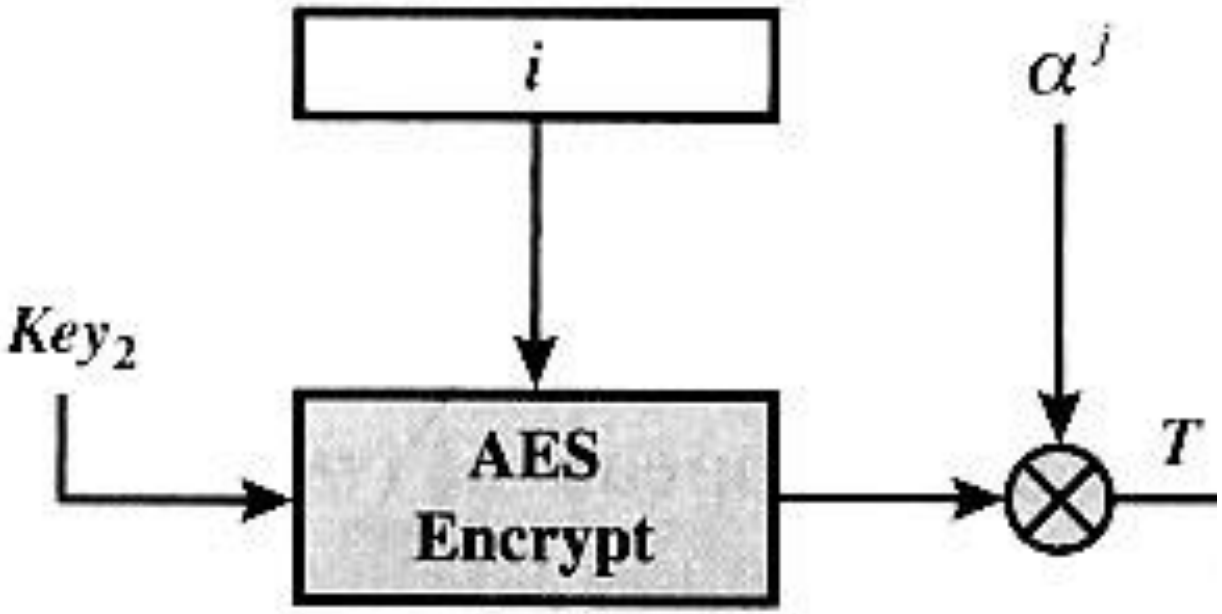
(a) Encryption

XTS-AES Decryption



(b) Decryption

Tweak Multiplied by α j times



$GF(2^{128})$

- 16 bytes
- addition: XOR corresponding bytes
- multiplying by 2 mod 0x100..0087:
  - left shift each byte by one position
  - most significant bit (msb, 0x80) of each byte becomes the least significant bit of the preceding byte (|= 1)
  - If first byte has had msb = 1, add 0x87 to the last byte.
- multiplying by 2^j mod n is the same as j rounds of multiplying by 2 mod n.

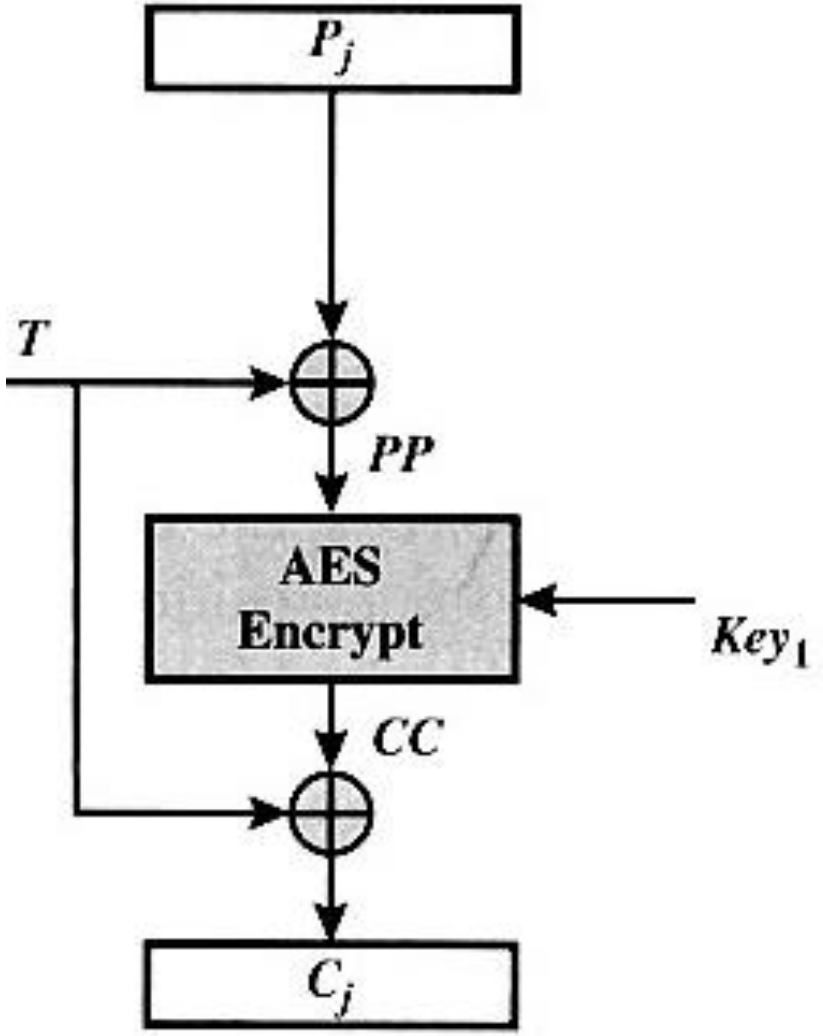
```
void readTweak(String filename){
  Scanner in = null;
  try {
    in = new Scanner(new File(filename));
  } catch (FileNotFoundException e){
    System.err.println(filename + " not found");
    System.exit(1);
  }
  String tweak = in.nextLine();
  in.close();
  for (int i = 0; i < blockSize; i++)
   state[i] = Integer.parseInt(hexkey.substring(i * 2, (i + 1) * 2), 16);
  blockCipher2();
  copyBlock(T, state);
}
```

Multiplying T by α (polynomial x) in GF(2^{128})

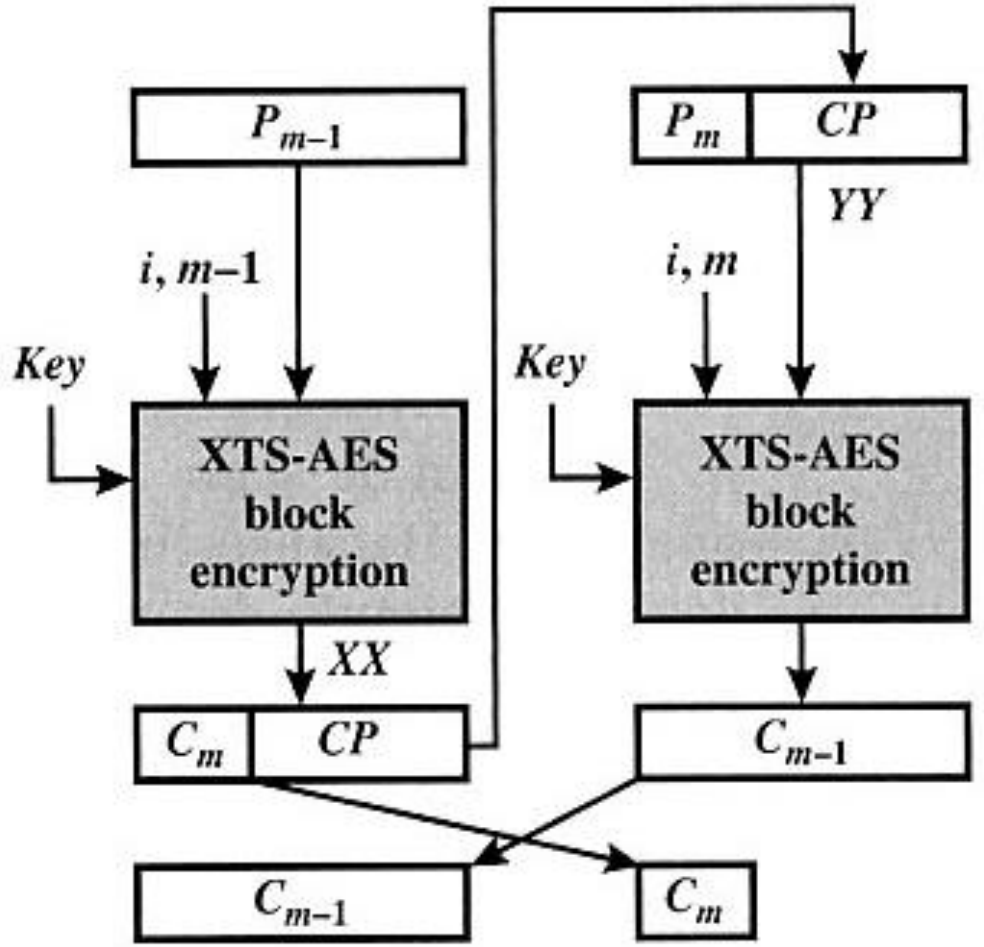
```
void Tx2(){
   boolean carry = ((T[0] & msb) != 0);
   for (int i = 0; i < blockSize - 1; i++){
     if ((T[i] & msb) != 0) T[i] ^= msb;
     T[i] <<= 1;
     if ((T[i + 1] & msb) != 0) T[i] |= 1;
   }
   if ((T[blockSize - 1] & msb) != 0) T[blockSize - 1] ^= msb;
   T[blockSize - 1] <<= 1;
   if (carry) T[blockSize - 1] ^= modulus;
 }
```

XOR-Encrypt-XOR (XEX)

```
void xex(){
    addBlock(state, T);
    blockCipher1();
    addBlock(state, T);
  }
```



XTS = XEX Tweakable Stealing



Stealing at the Final Block

XTS-AES mode with null final block	$C_j = \text{XTS-AES-blockEnc}(K, P_j, i, j) \quad j = 0, \ldots, m-1$
	$P_j = \text{XTS-AES-blockEnc}(K, C_j, i, j) \quad j = 0, \ldots, m-1$
XTS-AES mode with final block containing s bits	$C_j = \text{XTS-AES-blockEnc}(K, P_j, i, j) \quad j = 0, \ldots, m-2$ $XX = \text{XTS-AES-blockEnc}(K, P_{m-1}, i, m-1)$ $CP = \text{LSB}_{128-s}(XX)$ $YY = P_m \parallel CP$ $C_{m-1} = \text{XTS-AES-blockEnc}(K, YY, i, m)$ $C_m = \text{MSB}_s(XX)$
	$P_j = \text{XTS-AES-blockDec}(K, C_j, i, j) \quad j = 0, \ldots, m-2$ $YY = \text{XTS-AES-blockDec}(K, C_{m-1}, i, m-1)$ $CP = \text{LSB}_{128-s}(YY)$ $XX = C_m \parallel \text{CP}$ $P_{m-1} = \text{XTS-AES-blockDec}(K, XX, i, m)$ $P_m = \text{MSB}_s(YY)$

```
void encrypt(){
   int[] lastBlock = new int[blockSize];
   int len = readBlock();
   xex(); copyBlock(lastBlock, state); tx2();
   while ((len = readBlock()) >= 0){
     xex();
     if (len == blockSize){
       writeBlock(lastBlock, blockSize);
       copyBlock(lastBlock, state);
       tx2();
     }else{
        writeBlock(state, blockSize);
        writeBlock(lastBlock, len);
     }
   }
   System.out.flush();
 }
```

Homework 12: due 3-4-15

- H12CBCA.java, H12CFBA.java, H12OFBA.java, H12CTRA.java, H12CTSA.java, and H12XTSA.java are implementations of modes of operations CBC, CFB, OFB, CTR, CTS, and XTS of AES encryption.
- Implement the six AES modes of operations for decryption and test your programs on the given test12 encrypted files. If you think an encryption program is also the decryption one, no separate decryption program is needed.
- Submit your programs and test results.