Modern Cryptography
and Its Applications

5 Modes of Operation and RC4

Ch7 & Sec8.4-8.5 in textbook

Yanwei Yu

E-mail: ywyu@ustc.edu.cn

Outline

- Modes of Operation
- Random Bit Generation
- Stream Ciphers
  - RC4
- Purpose of Symmetric Encryption

Outline

- Modes of Operation
- Random Bit Generation
- Stream Ciphers
  - RC4
- Purpose of Symmetric Encryption

Learning Object

- Compare five modes of operation
  - How to work in encryption and decryption
  - Properties
  - Traditional application

Modes of Operation

- block ciphers encrypt fixed size blocks
  - eg. DES encrypts 64-bit blocks with 56-bit key
  - eg. AES encrypts 128-bit blocks with 128/192/256-bit key
- modes of operation: some way to en/decrypt arbitrary amounts of data in practise
  - 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.

- **ANSI :**
  - **defines 4 possible modes In FIPS 81** standard
  - **has expanded to 5 modes In 800-38A later**
  - **all modes are intended for use with any symmetric block cipher, including 3-DES and AES**
- **have** block **and** stream **modes**

Five Modes Defined In 800-38A

- Block modes: may need padding the last block
  - Electronic Codebook Book (ECB)
  - Cipher Block Chaining (CBC)

- Stream modes: Encryption function also used for decryption
  - Cipher Feedback (CFB)
  - Output Feedback (OFB)
  - Counter (CTR)

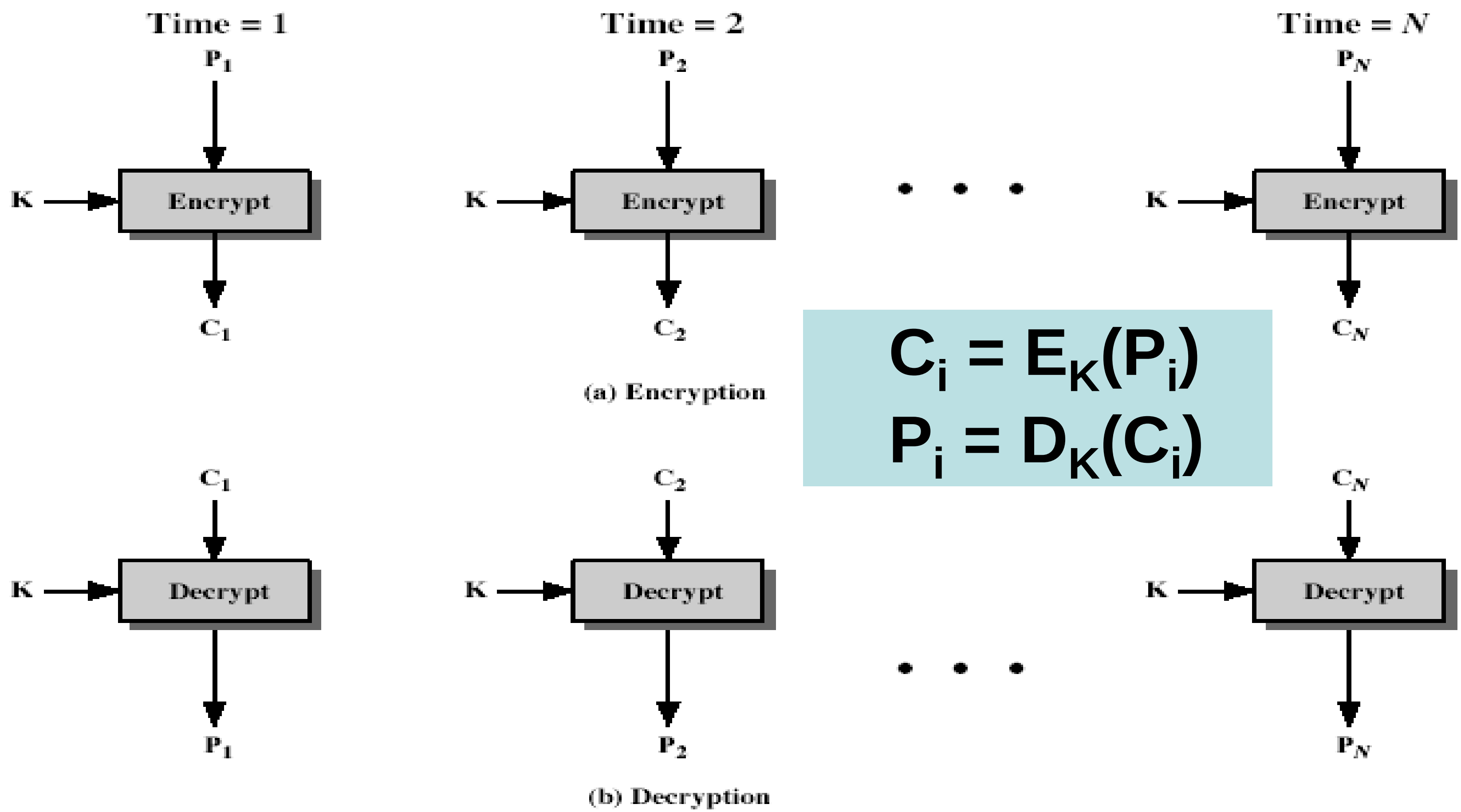
Electronic Codebook Book (ECB)

- Encrypting data directly with the block cipher
- **message is broken into independent blocks which are encrypted using the same key**

$$C_i = E_K(P_i)$$

- **Why use the term "codebook"?**
  - **for a given key k, there is a unique ciphertext for every possible block of plaintext.**

Electronic Codebook Book (ECB)



$$C_i = E_K(P_i)$$
$$P_i = D_K(C_i)$$

Example

Given $C_i = E_K(P_i)$ $(i \geq 1)$

$C'_i = E_K(P'_i)$ $(i \geq 1)$

$P = P_0 P_1 P_2 P_3 P_4 P_5 \ldots$; $C = C_0 C_1 C_2 C_3 C_4 C_5 \ldots$

$P^* = P_0 P_1 P_1 P_1 P_4 P_5 \ldots$

$P' = P_0 P_1 P'_2 P'_3 P'_4 P'_5 \ldots$

Then, $C^* = C_0 C_1 C_1 C_1 C_4 C_5 \ldots$

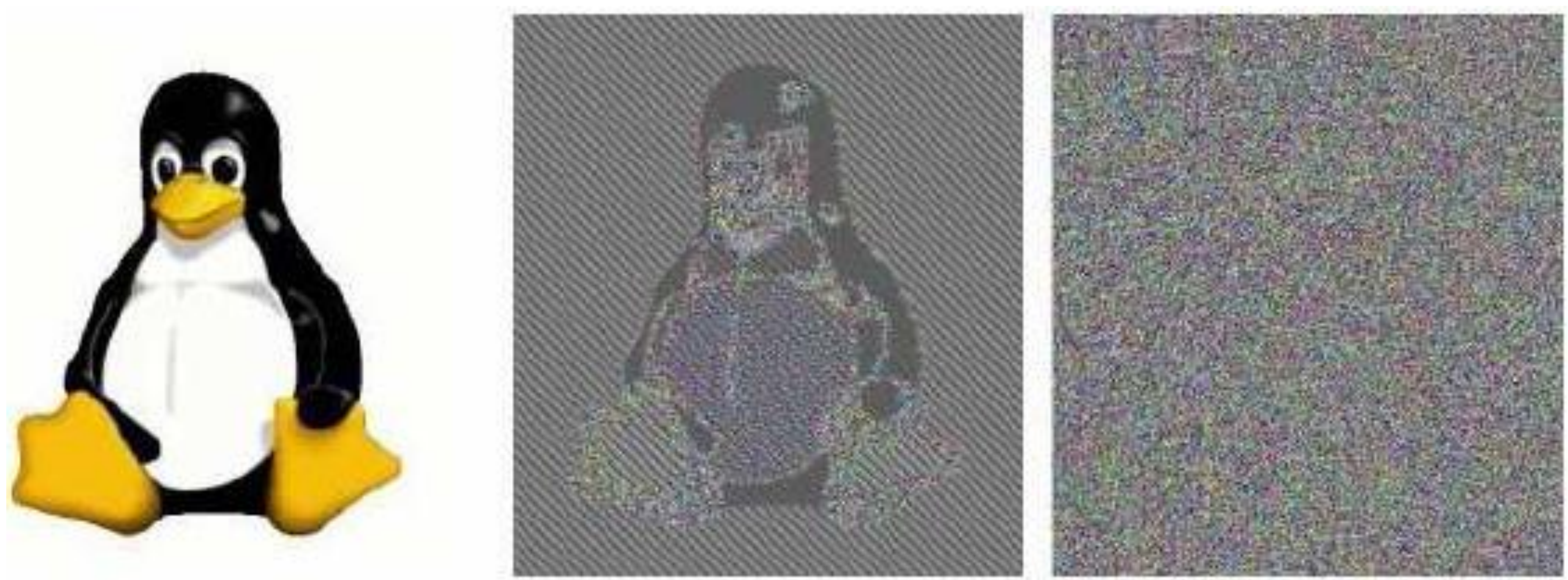
$C' = C_0 C_1 C'_2 C'_3 C'_4 C'_5 \ldots$

Properties of ECB

- Each message block is encrypted independently using the same key.
- Same plaintext blocks (under the same key) result in same ciphertext blocks.

- Derive:
  - message repetitions may show in ciphertext
  - Structure of message may show in ciphertext

Example of Image Encryption



Original   Encrypted using ECB mode   Encrypted securely

Handle Message length

- For ECB and CBC modes
- at end of message must handle a possible last short block
  - The last block may be not as large as blocksize of cipher
  - how to handle?

Handle Message length

- **two suggested (both equally valid) solutions**
  - ***ciphertext stealing*. pass the last ciphertext block through the cipher in ECB mode and XOR the output of that against the remaining message bytes (or bits). (recommend)**
  - ***Add zero bits*. pad the last block with enough zero bits to make the message length a proper multiple of the cipher block size.**
    - **may require an extra entire block over those in message**
    - **How to distinguish added bits and zero bits at end of message? (using separator or pad size)**
      - **eg. [b1 b2 b3 1 0 0 0 0]**
      - **[b1 b2 b3 0 0 0 0 5]** (ANSI X.923)
        - » **means have 3 data bytes, then 5 bytes zero-padding+count**
      - **[b1 b2 b3 5 5 5 5 5]** (PKCS#7)

Use of ECB

- Not suit for lengthy messages
  - due to the encrypted message blocks being independent

- uses: secure transmission of single values, e.g. IVs for CBC mode.

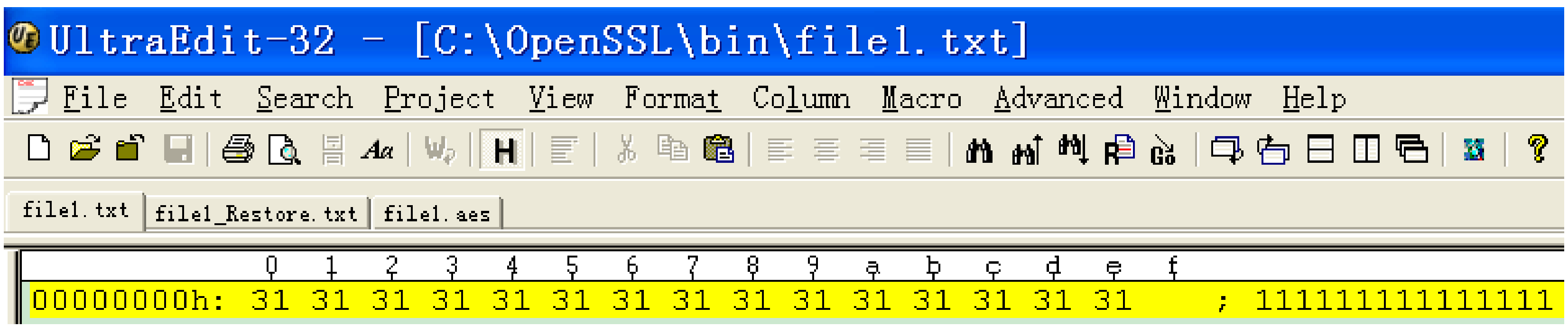
Use OpenSSL

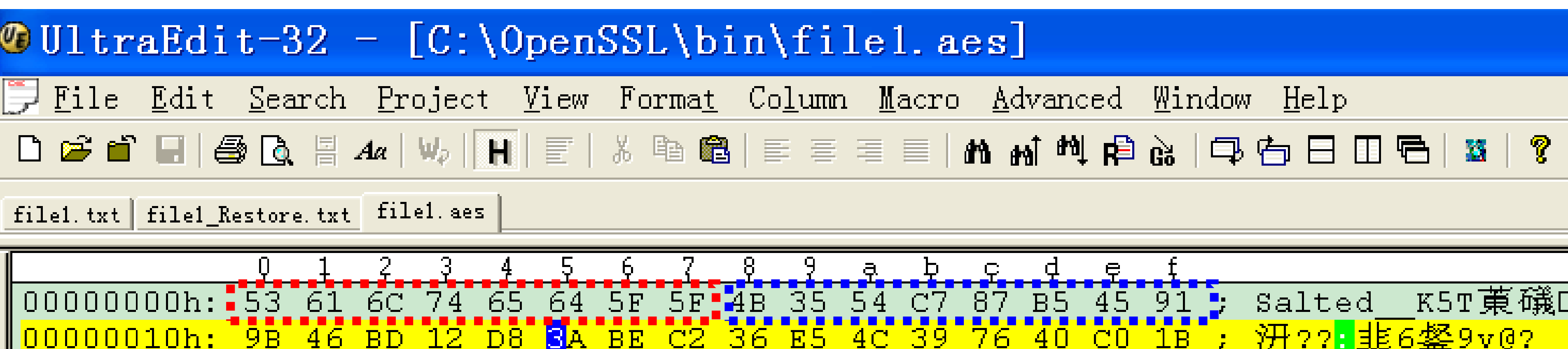
- **P1='111111111111111'   (15 Byte)**
- **Use aes-128-ecb**
- **Run openssl.exe**

```
OpenSSL> aes-128-ecb -in file1.txt -p -e -out file1.aes
enter aes-128-ecb encryption password:
Verifying - enter aes-128-ecb encryption password:
salt=4B3554C787B54591
key=8112A4A4413526372237B40ED3B8344B
iv =D0E686A008E0F380AE63423578E8C442
OpenSSL> aes-128-ecb -in file1.aes -p -d -out file1_Restore.txt
enter aes-128-ecb decryption password:
salt=4B3554C787B54591
key=8112A4A4413526372237B40ED3B8344B
iv =D0E686A008E0F380AE63423578E8C442
```

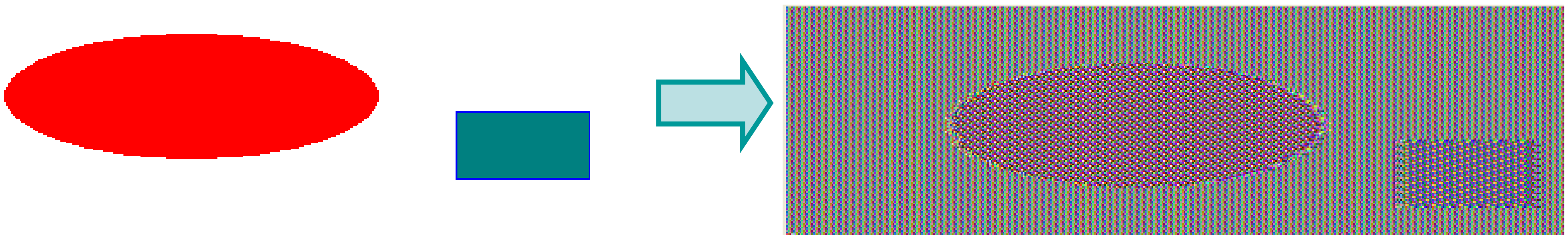
- **Open file1.txt , file1_Restore.txt and file1.aes using UltraEdit in hex**
- **Analyze the result**

```
OpenSSL> aes-128-ecb -in file1.txt -p -e -out file1.aes
enter aes-128-ecb encryption password:
Verifying - enter aes-128-ecb encryption password:
salt=4B3554C787B54591
key=8112A4A44135263722237B40ED3B8344B
iv =D0E686A008E0F380AE63423578E8C442
OpenSSL> aes-128-ecb -in file1.aes -p -d -out file1_Restore.txt
enter aes-128-ecb decryption password:
salt=4B3554C787B54591
key=8112A4A44135263722237B40ED3B8344B
iv =D0E686A008E0F380AE63423578E8C442
```

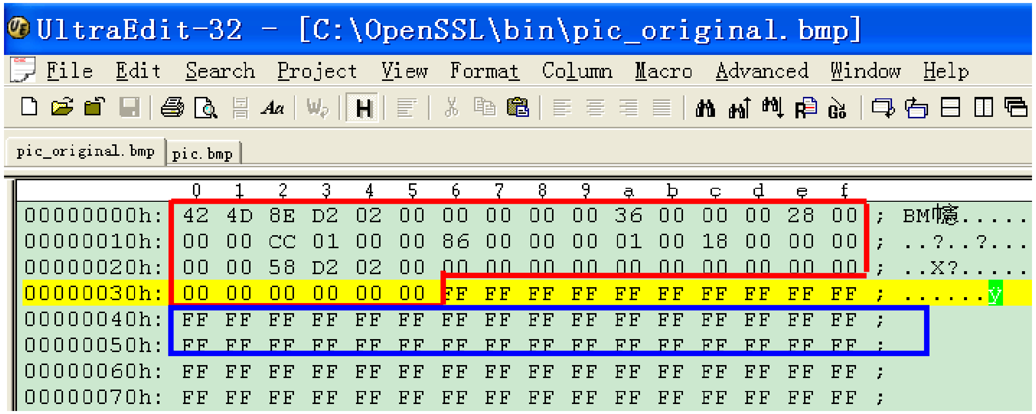


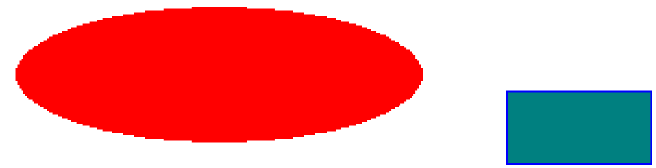


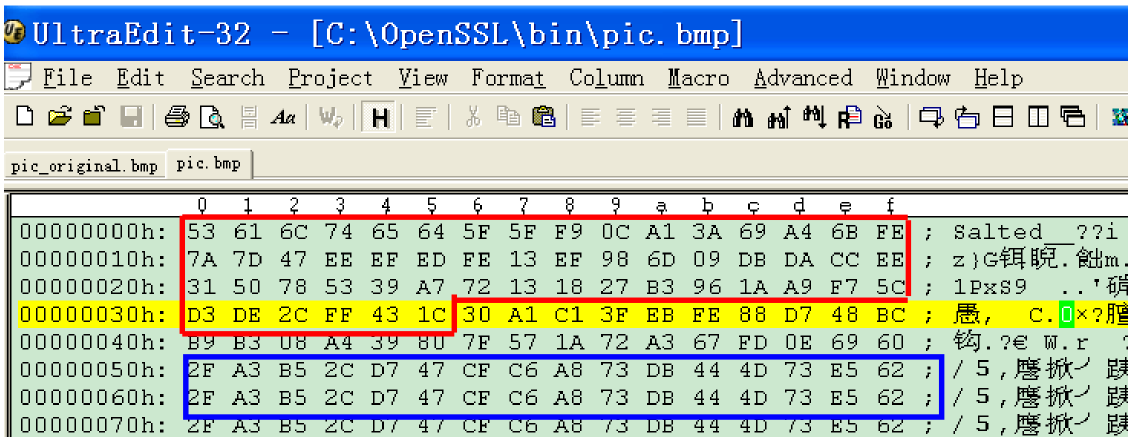
Enc BMP picuture using aes-128-ecb

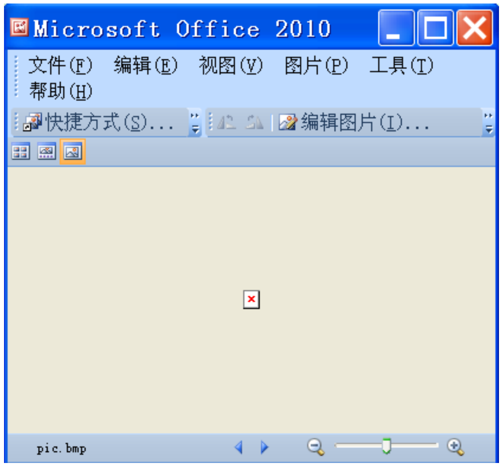


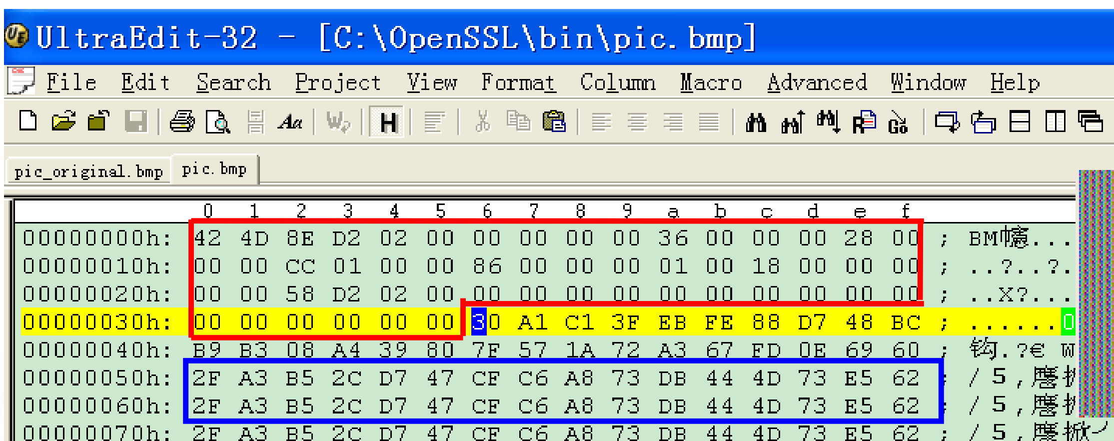
```
OpenSSL> aes-128-ecb -in pic_original.bmp -p -e -out pic.bmp
enter aes-128-ecb encryption password:
Verifying - enter aes-128-ecb encryption password:
salt=F90CA13A69A46BFE
key=49DAD74B857F0EBB4AB3CDFDAAFE071C
iv =54D9EEE77EB2EA4FAB8D00A1D4C19C86
```

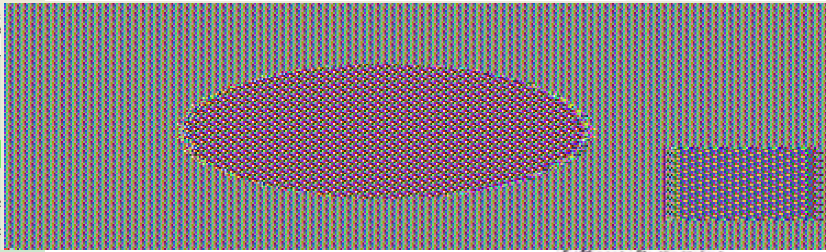

UltraEdit-32 - [C:\OpenSSL\bin\pic_original.bmp]
File Edit Search Project View Format Column Macro Advanced Window Help
pic_original.bmp pic.bmp
00000000h: 42 4D 8E D2 02 00 00 00 00 00 36 00 00 00 28 00
00000010h: 00 00 CC 01 00 00 86 00 00 00 01 00 18 00 00 00
00000020h: 00 00 58 D2 02 00 00 00 00 00 00 00 00 00 00 00
00000030h: 00 00 00 00 00 00 FF FF FF FF FF FF FF FF FF FF
00000040h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000050h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000060h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000070h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF



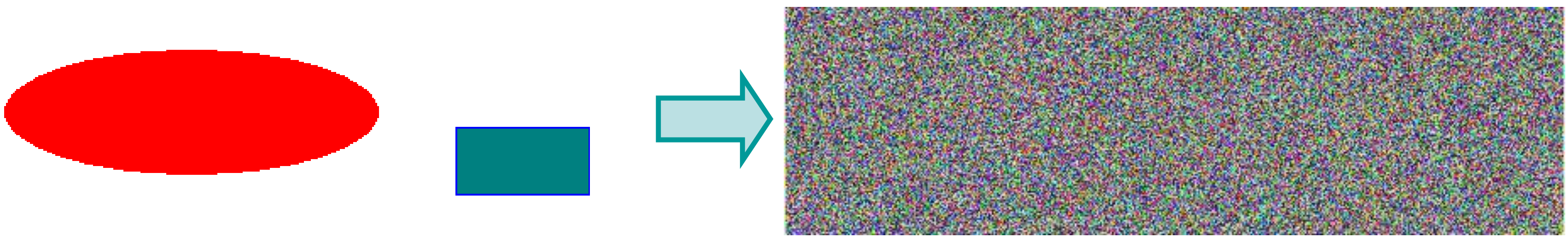

UltraEdit-32 - [C:\OpenSSL\bin\pic.bmp]
File Edit Search Project View Format Column Macro Advanced Window Help
pic_original.bmp pic.bmp
00000000h: 53 61 6C 74 65 64 5F 5F F9 0C A1 3A 69 A4 6B FE ; Salted__
00000010h: 7A 7D 47 EE EF ED FE 13 EF 98 6D 09 DB DA CC EE
00000020h: 31 50 78 53 39 A7 72 13 18 27 B3 96 1A A9 F7 5C
00000030h: D3 DE 2C FF 43 1C 30 A1 C1 3F EB FE 88 D7 48 BC
00000040h: B9 B3 08 A4 39 80 7F 57 1A 72 A3 67 FD 0E 69 60
00000050h: 2F A3 B5 2C D7 47 CF C6 A8 73 DB 44 4D 73 E5 62
00000060h: 2F A3 B5 2C D7 47 CF C6 A8 73 DB 44 4D 73 E5 62
00000070h: 2F A3 B5 2C D7 47 CF C6 A8 73 DB 44 4D 73 E5 62


Microsoft Office 2010
文件(F) 编辑(E) 视图(V) 图片(P) 工具(T)
帮助(H)
快捷方式(S)...
编辑图片(I)...
pic.bmp

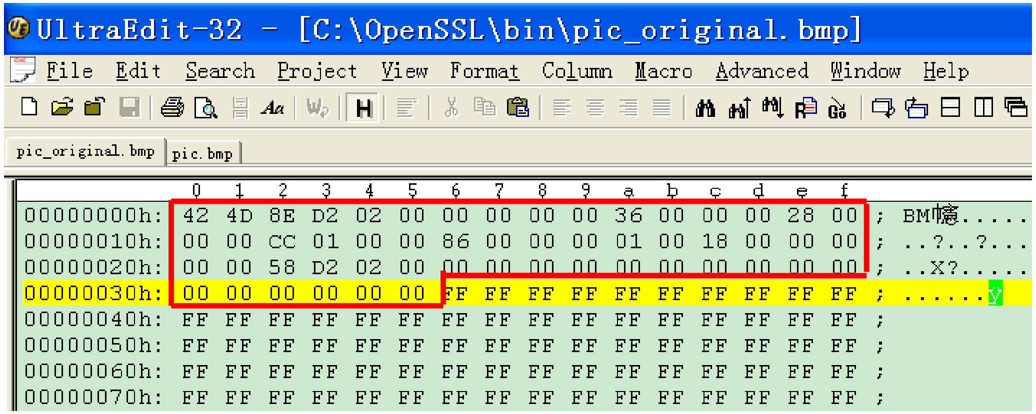

UltraEdit-32 - [C:\OpenSSL\bin\pic.bmp]
File Edit Search Project View Format Column Macro Advanced Window Help
pic_original.bmp pic.bmp
00000000h: 42 4D 8E D2 02 00 00 00 00 00 36 00 00 00 28 00
00000010h: 00 00 CC 01 00 00 86 00 00 00 01 00 18 00 00 00
00000020h: 00 00 58 D2 02 00 00 00 00 00 00 00 00 00 00 00
00000030h: 00 00 00 00 00 00 30 A1 C1 3F EB FE 88 D7 48 BC
00000040h: B9 B3 08 A4 39 80 7F 57 1A 72 A3 67 FD 0E 69 60
00000050h: 2F A3 B5 2C D7 47 CF C6 A8 73 DB 44 4D 73 E5 62
00000060h: 2F A3 B5 2C D7 47 CF C6 A8 73 DB 44 4D 73 E5 62
00000070h: 2F A3 B5 2C D7 47 CF C6 A8 73 DB 44 4D 73 E5 62

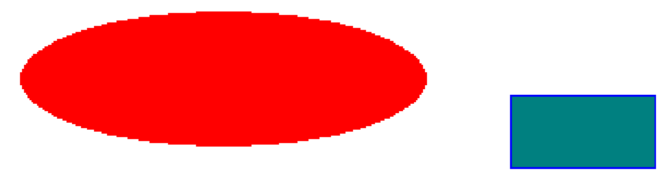


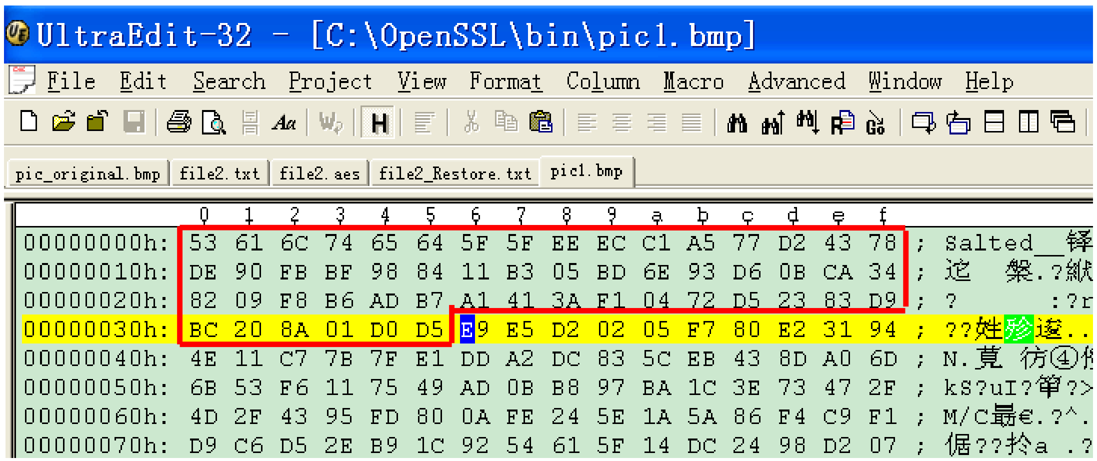
Enc BMP picuture using aes-128-cbc

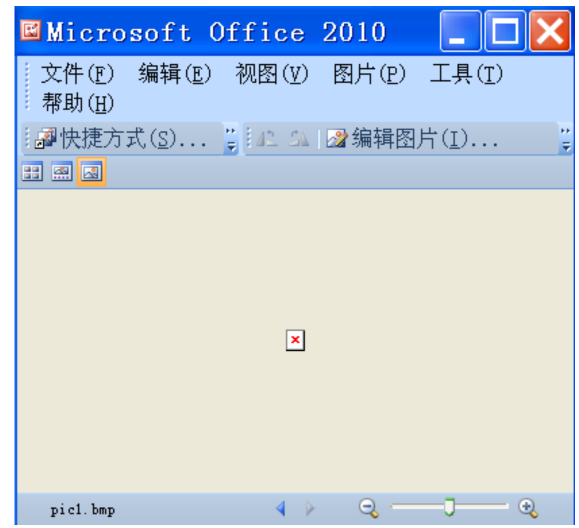


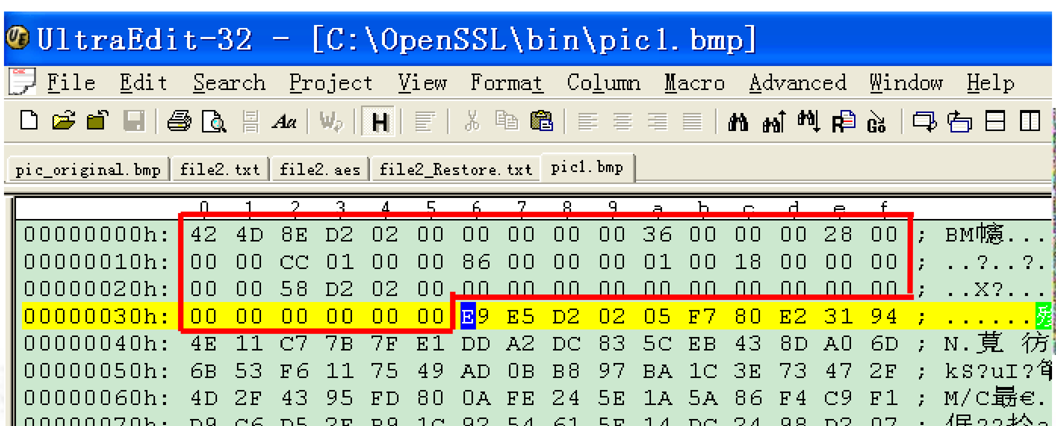
```
OpenSSL> aes-128-cbc -in pic_original.bmp -p -e -out pic1.bmp
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
salt=EEECC1A577D24378
key=B1190F9389AE050F728C1AA6CC5007B9
iv =FA60B32B507169EA36A530025E4A60F5
```


UltraEdit-32 - [C:\OpenSSL\bin\pic_original.bmp]
File Edit Search Project View Format Column Macro Advanced Window Help
pic_original.bmp pic.bmp
0 1 2 3 4 5 6 7 8 9 a b c d e f
00000000h: 42 4D 8E D2 02 00 00 00 00 00 36 00 00 00 28 00
00000010h: 00 00 CC 01 00 00 86 00 00 00 01 00 18 00 00 00
00000020h: 00 00 58 D2 02 00 00 00 00 00 00 00 00 00 00 00
00000030h: 00 00 00 00 00 00 FF FF FF FF FF FF FF FF FF FF
00000040h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000050h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000060h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000070h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF




UltraEdit-32 - [C:\OpenSSL\bin\pic1.bmp]
File Edit Search Project View Format Column Macro Advanced Window Help
pic_original.bmp file2.txt file2.aes file2_Restore.txt pic1.bmp
0 1 2 3 4 5 6 7 8 9 a b c d e f
00000000h: 53 61 6C 74 65 64 5F 5F EE EC C1 A5 77 D2 43 78 ; Salted__
00000010h: DE 90 FB BF 98 84 11 B3 05 BD 6E 93 D6 0B CA 34
00000020h: 82 09 F8 B6 AD B7 A1 41 3A F1 04 72 D5 23 83 D9
00000030h: BC 20 8A 01 D0 D5 E9 E5 D2 02 05 F7 80 E2 31 94
00000040h: 4E 11 C7 7B 7F E1 DD A2 DC 83 5C EB 43 8D A0 6D
00000050h: 6B 53 F6 11 75 49 AD 0B B8 97 BA 1C 3E 73 47 2F
00000060h: 4D 2F 43 95 FD 80 0A FE 24 5E 1A 5A 86 F4 C9 F1
00000070h: D9 C6 D5 2E B9 1C 92 54 61 5F 14 DC 24 98 D2 07


Microsoft Office 2010
文件(F) 编辑(E) 视图(V) 图片(P) 工具(T)
帮助(H)
快捷方式(S)...
编辑图片(I)...
pic1.bmp


UltraEdit-32 - [C:\OpenSSL\bin\pic1.bmp]
File Edit Search Project View Format Column Macro Advanced Window Help
pic_original.bmp file2.txt file2.aes file2_Restore.txt pic1.bmp
0 1 2 3 4 5 6 7 8 9 a b c d e f
00000000h: 42 4D 8E D2 02 00 00 00 00 00 36 00 00 00 28 00
00000010h: 00 00 CC 01 00 00 86 00 00 00 01 00 18 00 00 00
00000020h: 00 00 58 D2 02 00 00 00 00 00 00 00 00 00 00 00
00000030h: 00 00 00 00 00 00 E9 E5 D2 02 05 F7 80 E2 31 94
00000040h: 4E 11 C7 7B 7F E1 DD A2 DC 83 5C EB 43 8D A0 6D
00000050h: 6B 53 F6 11 75 49 AD 0B B8 97 BA 1C 3E 73 47 2F
00000060h: 4D 2F 43 95 FD 80 0A FE 24 5E 1A 5A 86 F4 C9 F1
00000070h: D9 C6 D5 2E B9 1C 92 54 61 5F 14 DC 24 98 D2 07

Criteria for evaluating

- **Overhead（总体比较）:**
  - additional operations for the encryption and decryption operation, when compared to ECB mode.
- **Error recovery: focus on the resynchronizes**
  - an transmission error in the i^{th} cipher-text block is inherited by only a few plaintext blocks after which the mode resynchronizes.

Criteria for evaluating

- **Error propagation:**
  - The property that an transmission error in the i^{th} cipher-text block is inherited by the i^{th} and all subsequent plaintext blocks.
- **Diffusion:**
  - How the plaintext statistics are reflected in the cipher-text
- **Security:**
  - Whether or not the cipher-text blocks leak information about the plaintext blocks.

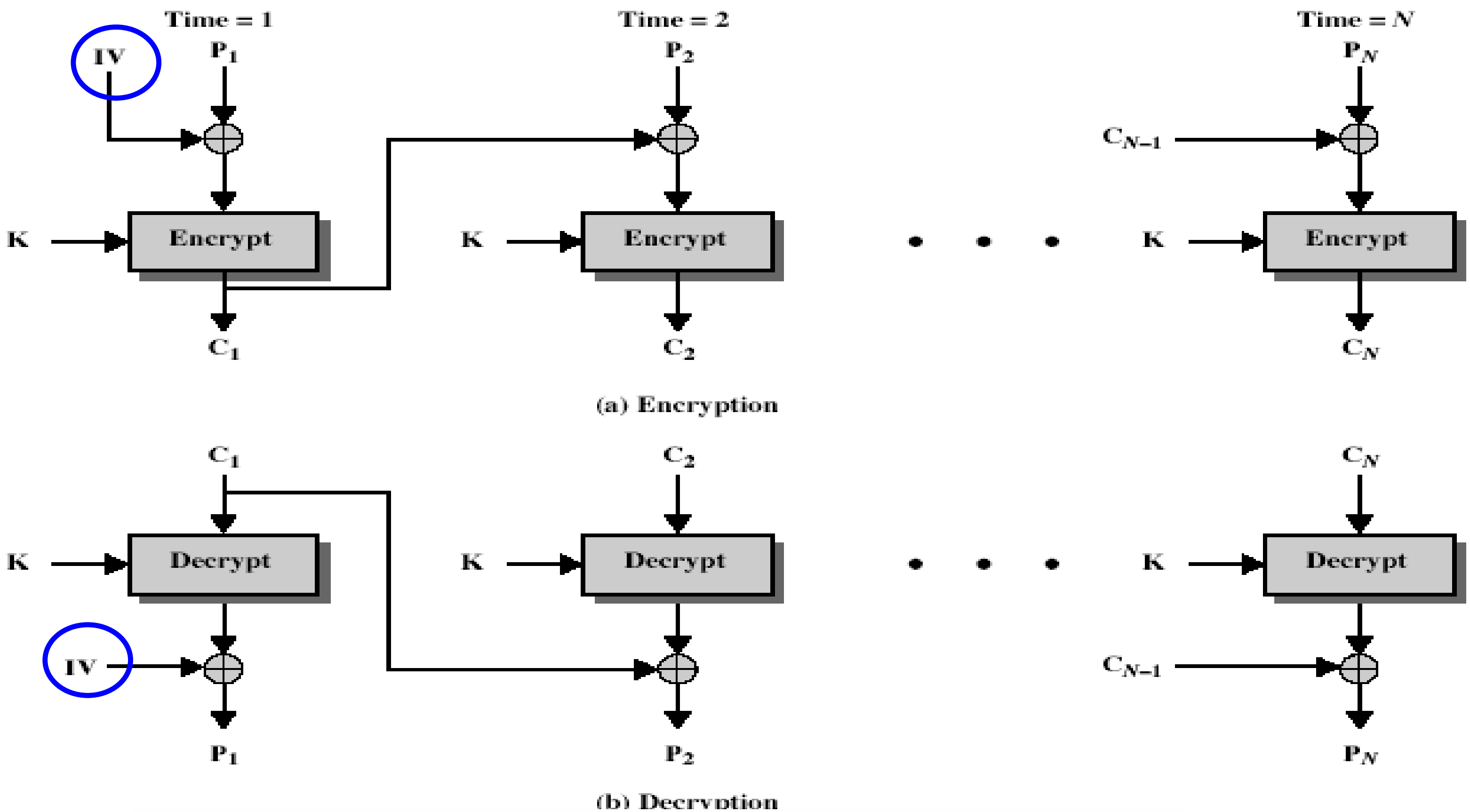
Cipher Block Chaining (CBC)

- message is broken into blocks
- Each block linked together in encryption operation to overcome the weakness of ECB
- each previous cipher blocks is chained with current plaintext block, hence name
- use Initial Vector (IV) to start process

$$C_i = E_K(P_i \text{ XOR } C_{i-1})\ (i \geqq 1)$$

$$C_0 = IV$$

Cipher Block Chaining (CBC)



(a) Encryption

(b) Decryption

$$C_0 = IV$$

$$C_i = E_K(P_i \text{ XOR } C_{i-1}) \ (i \geqq 1)$$

$$P_i = D_K(C_i) \text{ XOR } C_{i-1} \ (i \geqq 1)$$

Example

$C_0 = IV$

$C_i = E_K(P_i \text{ XOR } C_{i-1}) \ (i \geq 1)$

$P_i = D_K(C_i) \text{ XOR } C_{i-1} \ (i \geq 1)$

$C = C_1C_2C_3C_4C_5 \ldots$

$P = P_1P_2P_3P_4P_5 \ldots$

Case1:

$C' = C_1C'_2C_3C_4C_5C_6 \ldots$

$P' = P_1P'_2P'_3P_4P_5P_6 \ldots$

Case2:

$C' = C_1C_2C_2C_3C_4C_5C_6 \ldots$

$P' = P_1P_2P'_2P_3P_4P_5 \ldots$

Case3:

$C' = C_1C_2C_4C_5C_6 \ldots$

$P' = P_1P_2P'_4P_5P_6 \ldots$

Q:

$C' = C_1C'_2C'_3C'_4C_5C_6C_7C_8 \ldots$

$P' = P_1P'_2P'_3P'_4P'_5P_6P_7P_8 \ldots$

Characteristic of CBC

- a ciphertext block depends on all plaintext blocks before it
  - $C_i = E_K(P_i \text{ XOR } C_{i-1}) = E_K(P_i \text{ XOR } E_K(P_{i-1} \text{ XOR } C_{i-2})) =$ 。。。
- any change to a plaintext block affects all following ciphertext blocks (diffusion)
  - $C_i = E_K(P_i \text{ XOR } C_{i-1})$
  - $C_{i+1} = E_K(P_{i+1} \text{ XOR } E_K(P_i \text{ XOR } C_{i-1}))$
  - 。。。
- Error propagation(传播): a single bit error in ciphertext block C_j affects decipherment of blocks C_j and C_{j+1}
- Self-synchronizing: if an error occurs in block C_j but not C_{j+1}, C_{j+2} is correctly decrypted to P_{j+2}.
- Message Padding: same as ECB mode

IV (Initialization Vector) for CBC

- **What is IV for CBC mode?**
  - a data block that is that same size as the cipher block
  - XORed with the first block of plaintext to produce the first block of ciphertext on encryption
  - XORed with the output of the decryption algorithm to recover the first block of plaintext on decryption.
  - known to both the sender and receiver

$$C_1 = E(K, [IV \oplus P_1])$$

$$P_1 = IV \oplus D(K, C_1)$$

- **How to select IV for CBC mode?**
  - Random generation
    - If the IV and key are unchanged for different messages, CBC mode is reduced to ECB mode.
  - Integrity： should be protected against unauthorized changes, why?

$$P_1[i] = IV[i] \oplus D(K, C_1)[i]$$

$$P_1[i]' = IV[i]' \oplus D(K, C_1)[i]$$

- Two common ways to deal with the storage (or transmission) of the IV in CBC mode.
  - (simplest) generate an IV at random and the IV can be sent encrypted in ECB mode before rest of message.
  - (common) make it during the key negotiation(协商) process. E.g. derive both encryption keys and chaining IVs from a randomly generated shared secret in PKCS #5.

glance at PKCS #5

- **PKCS #5**
  - **(ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs5v2-0.pdf) is a standard put forth by the former RSA Data Security Corporation as one of a series of standards of public key cryptography.**
  - **It defines two KDF (Key derivation functions) algorithms called PBKDF1 and PBKDF2.**
- **PBKDF1 is an old standard that was originally meant to be used with DES, and as such has fixed size inputs.**
- **PBKDF2 is more flexible and the recommended KDF from the PKCS #5 standard**

Figure 5.6 PKCS #5 PBKDF2 Algorithm

Input:

secret:	The secret used as a *master key* to derive into a session key
salt:	The random nonsecret string used to prevent dictionary attacks
iterations	The number of iterations in the main loop
w:	The digest size of the hash being used
outlen:	The desired amount of KDF data requested

Output:

out:	The KDF data

1. for blkNo from 1 to ceil(*outlen*/*w*) do
    1. T = HMAC(*secret*, *salt* || blkNo)
    2. U = T
    3. for i from 1 to *iterations* do
        - i. T = HMAC(*secret*, T)
        - ii. U = U xor T
    4. *out* = *out* || U
2. Truncate *out* to *outlen* bytes

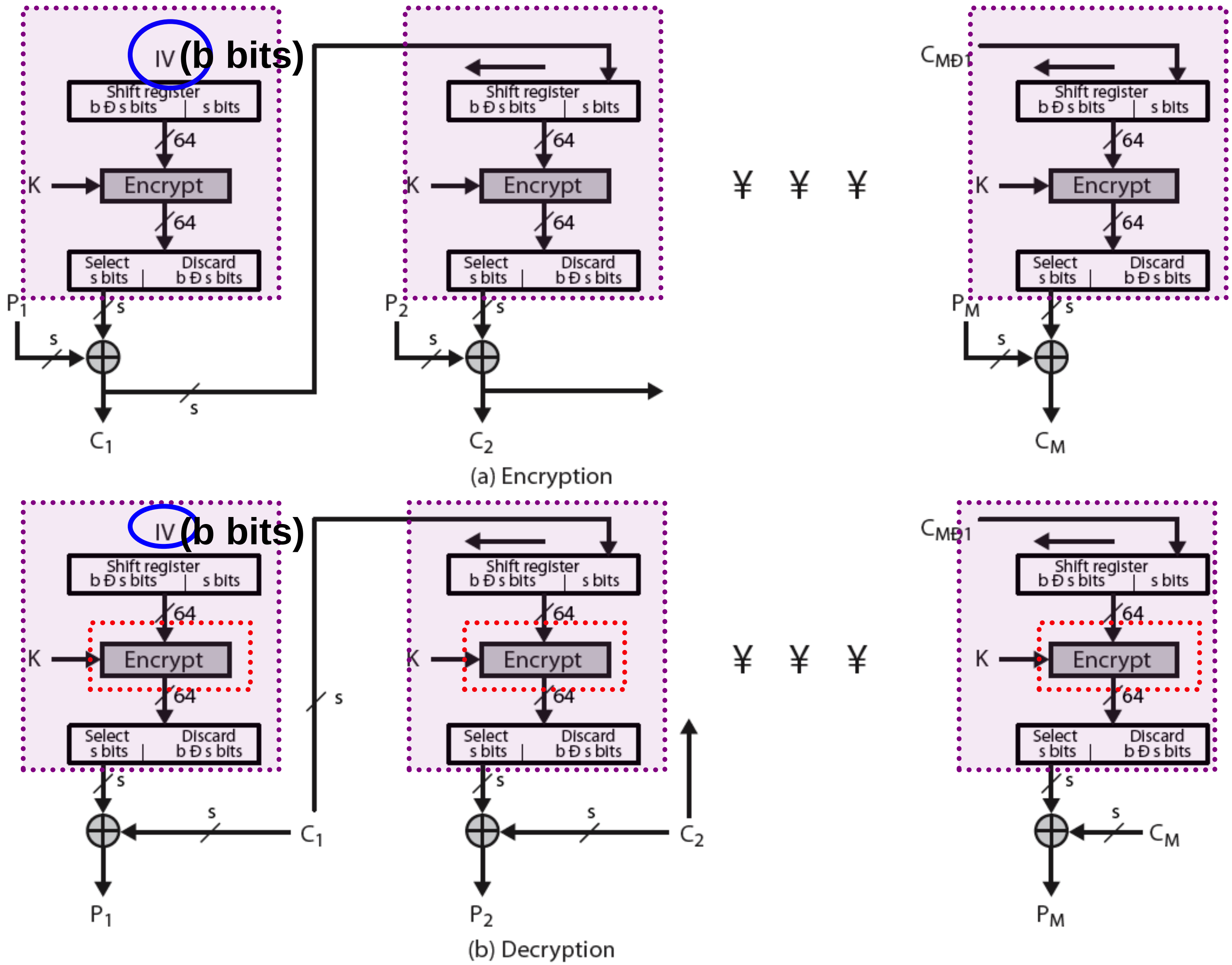
Use of CBC

- Bulk data encryption
  - General-purpose block-oriented transmission
- Authentication?
  - Why right? “a change in the ciphertext will make a nontrivial(不小的) change in the plaintext.” It is true that changing a single bit of ciphertext will alter two blocks of plaintext.
  - Myths in fact: easy to fabricate(伪造)
    - CMAC(CBC-MAC)

Cipher FeedBack (CFB)

- message is treated as a stream of bits
  - Some applications require that s-bit plaintext units be encrypted and transmitted without delay (often s = 1 or s = 8).
  - Not necessary to pad
- Message is added to the key stream (the output of the block cipher here)
- ciphertext are feed back for next stage (hence name)
- standard allows any number of bit (s=1,8, 64 or 128 etc) to be feed back
  - denoted CFB-1, CFB-8, CFB-64, CFB-128 etc
- most efficient to use all bits in block (64 or 128)
  - CFB-64, CFB-128

Cipher FeedBack (CFB)



DES: b=64, AES: b=128.

plaintext is divided into segments of s bits, rather than units of b bits. (b≥s)

Example

Use CFB-8 for DES

It means that b=64bits and S=8bits

$P_i=f(C_i,C_{i-1},\ldots,C_{i-b/s})$

$C=C_1C_2C_3C_4C_5\ \ldots$

$P=P_1P_2P_3P_4P_5\ \ldots$

Case1:

$C'=C_1C'_2C_3C_4C_5C_6\ \ldots$

$P'=P_1P'_2P'_3P'_4P'_5P'_6P'_7P'_8P'_9P'_{10}P_{11}\ \ldots$

Case2:

$C'=C_1C_2C_2C_3C_4C_5C_6\ \ldots$

$P'=P_1P_2P'_2P'_3P'_4P'_5P'_6P'_7P'_8P'_9P'_{10}P_{10}\ \ldots$

Case3:

$C'=C_1C_2C_4C_5C_6\ \ldots$

$P'=P_1P_2P'_4P'_5P'_6P'_7P'_8P'_9P'_{10}P'_{11}P_{12}\ldots$

Q:

$C'=C_1C'_2C'_3C'_4C_5C_6C_7C_8\ \ldots$

$P'=P_1P'_2P'_3P'_4P'_5P'_6P'_7P'_8P'_9P'_{10}P'_{11}P'_{12}P_{13}\ \ldots$

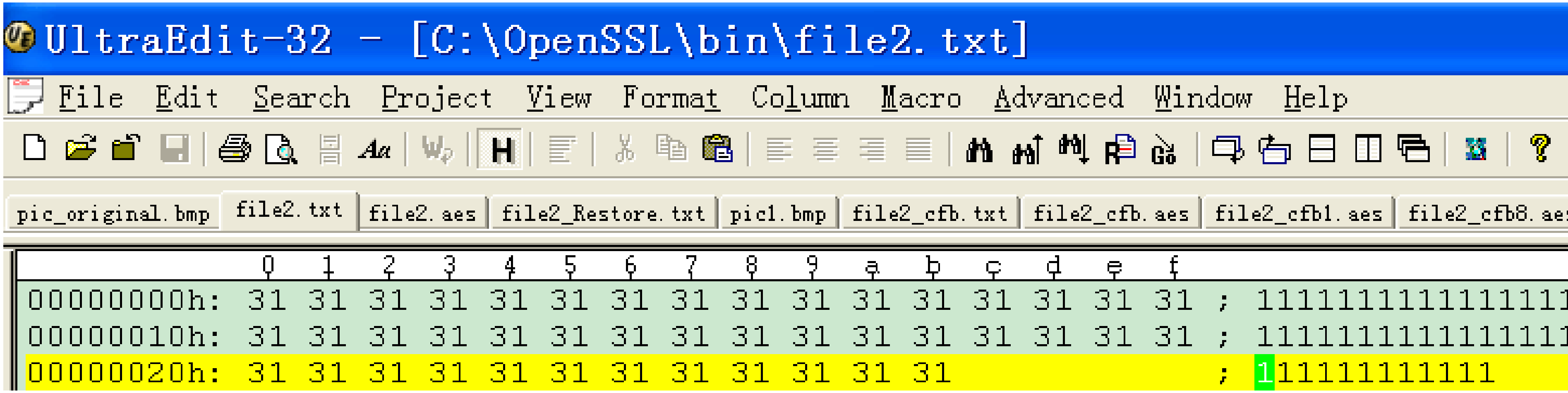
Characteristic of CFB

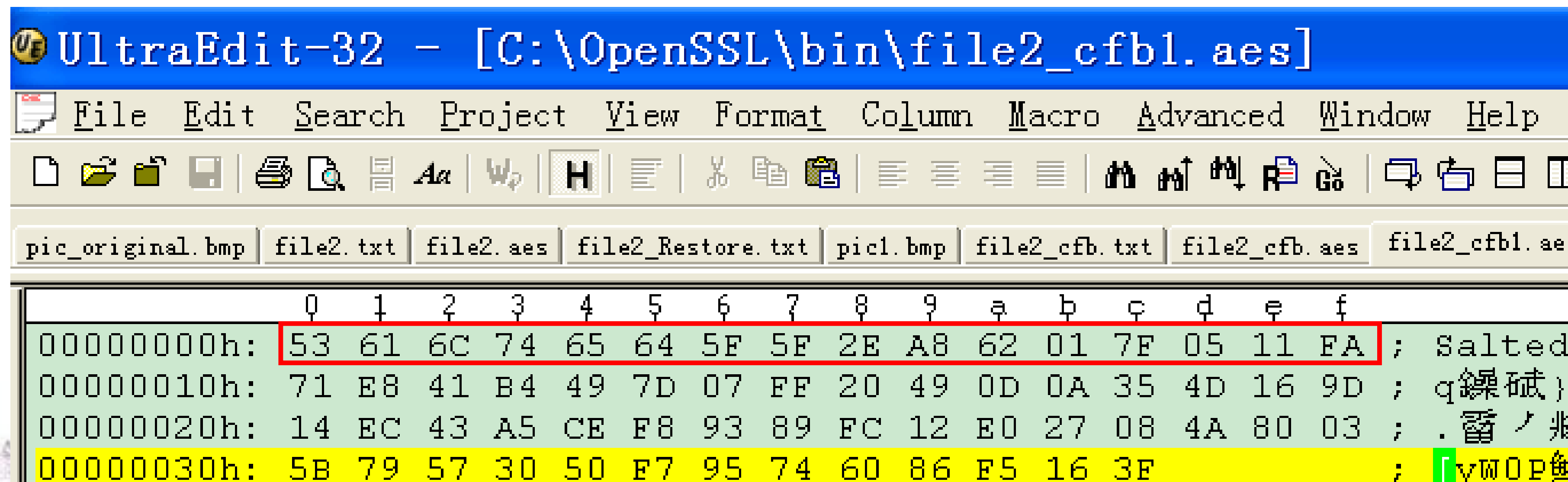
- **appropriate when data arrives in bits/bytes**
- **note that the block cipher is used in** encryption **mode at** both **ends(namely sender and receiver)**
- **errors propagate for several blocks after the error**
- **must** never **reuse the same key stream sequence (namely, key+IV must not repeat)**

Use of CFB

- stream data encryption
  - Some applications require that s-bit plaintext units be encrypted and transmitted without delay (often s = 1 or s = 8).
  - E.g. pay per view
- authentication

Using aes-128-cfb1

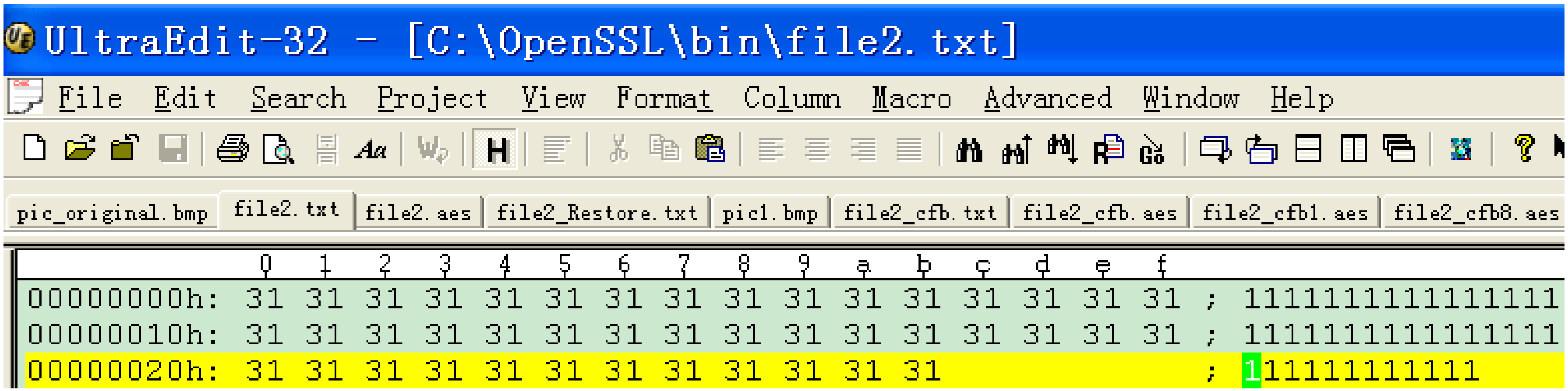


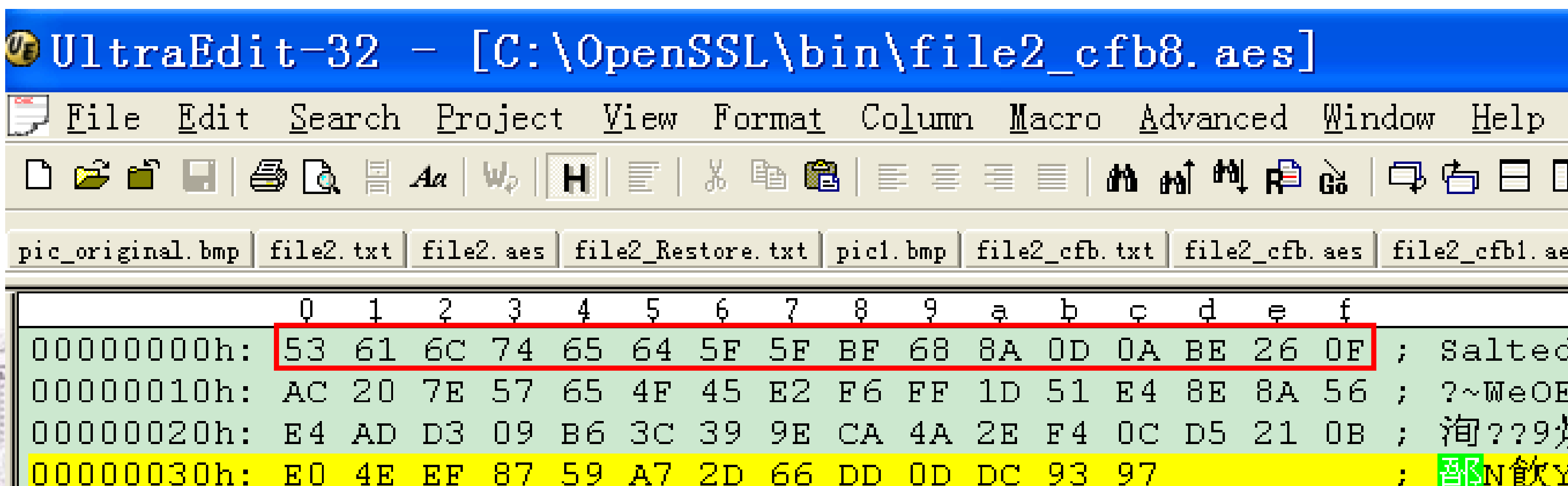


```
OpenSSL> enc -aes-128-cfb1 -in file2.txt -p -e -out file2_cfb1.aes
enter aes-128-cfb1 encryption password:
Verifying - enter aes-128-cfb1 encryption password:
salt=2EA862017F0511FA
key=67A3F57DC70DAEEE19B53140A37C2F3A
iv =E91CDA35762BBCA8E6FB479872561E41
```

```
OpenSSL> enc -aes-128-cfb1 -in file2_cfb1.aes -p -d -out file2_cfb81.txt
enter aes-128-cfb1 decryption password:
salt=2EA862017F0511FA
key=67A3F57DC70DAEEE19B53140A37C2F3A
iv =E91CDA35762BBCA8E6FB479872561E41
```

Using aes-128-cfb8





```
OpenSSL> enc -aes-128-cfb8 -in file2.txt -p -e -out file2_cfb8.aes
enter aes-128-cfb8 encryption password:
Verifying - enter aes-128-cfb8 encryption password:
salt=BF688A0ABE260FAC
key=645D13FF3D11C38D3AF31237699B82A3
iv =ECE8154FAF53A3DF4587E389388D4657
```

```
OpenSSL> enc -aes-128-cfb8 -in file2_cfb8.aes -p -d -out file2_cfb8.txt
enter aes-128-cfb8 decryption password:
salt=BF688A0ABE260FAC
key=645D13FF3D11C38D3AF31237699B82A3
iv =ECE8154FAF53A3DF4587E389388D4657
```

Output FeedBack (OFB)

- message is treated as a stream of bits
- Message is added to the key stream (the output of the block cipher here)
- output(namely the key stream) is then feed back (hence name)
- feedback is independent of message
  - Can preprocess to generate key stream in advance of need to enhance throughput
- can be computed in advance

$$C_i = P_i \text{ XOR } O_i \quad (i \geq 1)$$

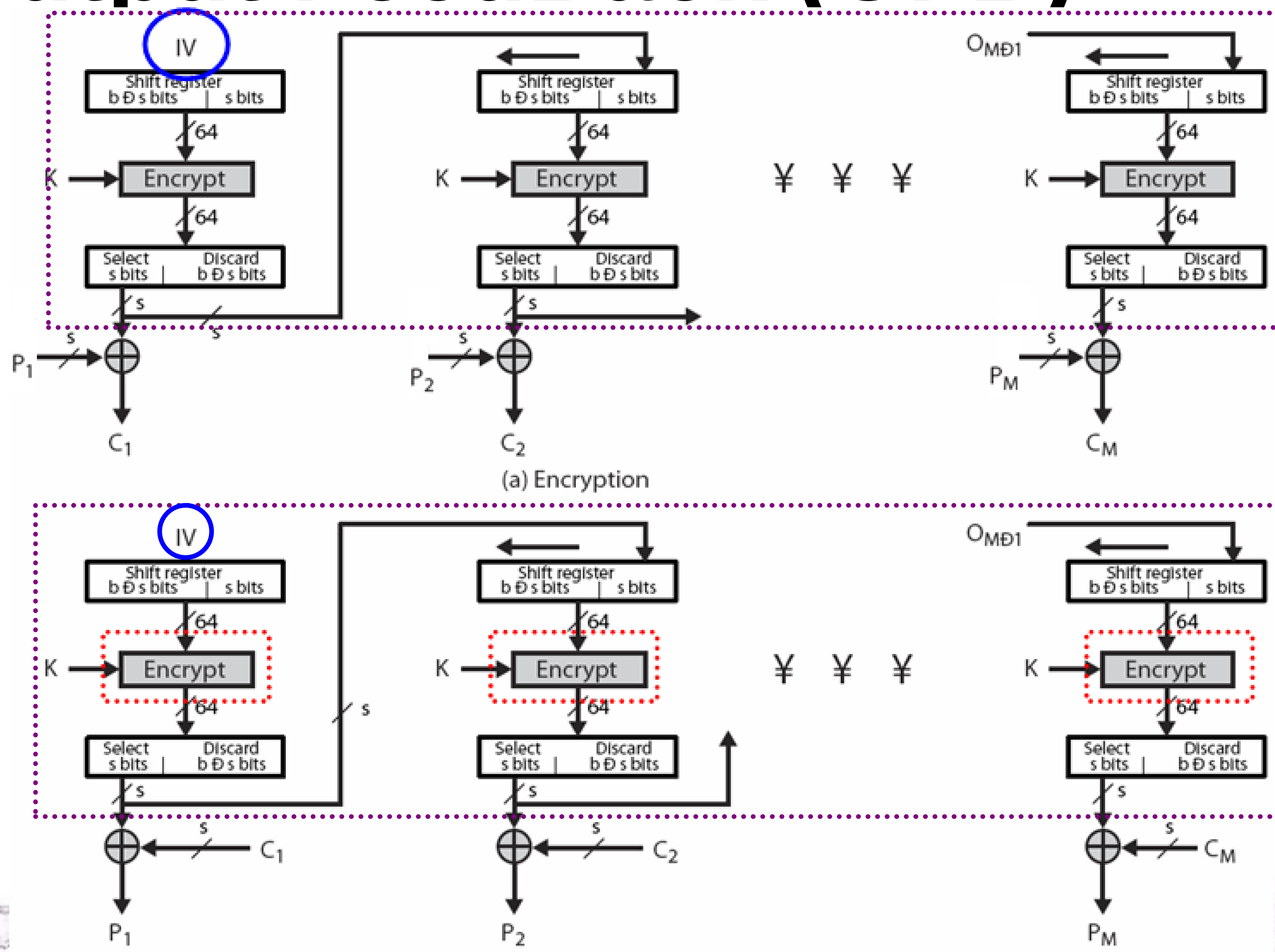
$$O_i = DES_{K1}(O_{i-1}, \ldots, O_{i-b/s}) \quad (i \geq 1)$$

$$O_0 = IV$$

- uses: stream encryption on noisy channels

Output FeedBack

$$C_i = P_i \text{ XOR } O_i \quad (i \geq 1)$$
$$O_i = DES_{K1}(O_{i-1}, \ldots, O_{i-b/s}) \quad (i \geq 1)$$
$$O_0 = IV$$



(a) Encryption

(b) Decryption

Example

$$C_i = P_i \text{ XOR } O_i \quad (i \geq 1)$$
$$O_i = DES_{K1}(O_{i-1}, \ldots, O_{i-b/s}) \quad (i \geq 1)$$
$$O_0 = IV$$

Use OFB-8 for DES

It means that b=64bits and S=8bits

$P_i = f(C_i, O_{i-1}, \ldots, O_{i-b/s})$

$C = C_1C_2C_3C_4C_5 \ \ldots$

$P = P_1P_2P_3P_4P_5 \ \ldots$

Case1:

$C' = C_1C'_2C_3C_4C_5C_6 \ \ldots$

$P' = P_1P'_2P_3P_4P_5P_6 \ \ldots$

Case2:

$C' = C_1C_2C_2C_3C_4C_5C_6 \ \ldots$

$P' = P_1P_2P'_3P'_4P'_5P'_6P'_7 \ \ldots$

Case3:

$C' = C_1C_2C_4C_5C_6 \ \ldots$

$P' = P_1P_2P'_3P'_4P'_5 \ldots$

Q:

$C' = C_1C'_2C'_3C'_4C_5C_6C_7C_8 \ \ldots$

$P' = P_1P'_2P'_3P'_4P_5P_6P_7P_8 \ \ldots$

OFB (vs. CFB)

- **bit errors do not propagate**
- **uses: stream encryption on noisy channels**

- **more vulnerable to message stream modification attack than CFB**
  - **complementing(取反) a bit in the ciphertext complements the corresponding bit in the recovered plaintext**
  - $P_i = C_i \text{ XOR } O_i$, $O_i = E_k(O_{i-1}, \ldots, O_{i-b/s})$
- **must** never **reuse the same key stream sequence (namely, key+IV must not repeat)**
  - **Similar to CFB**
- **sender & receiver must remain in sync**

Counter (CTR)

- a “new” mode, though proposed early on
- similar to OFB but encrypts counter value rather than any feedback value
- must have a different key & counter value for every plaintext block (never reused)

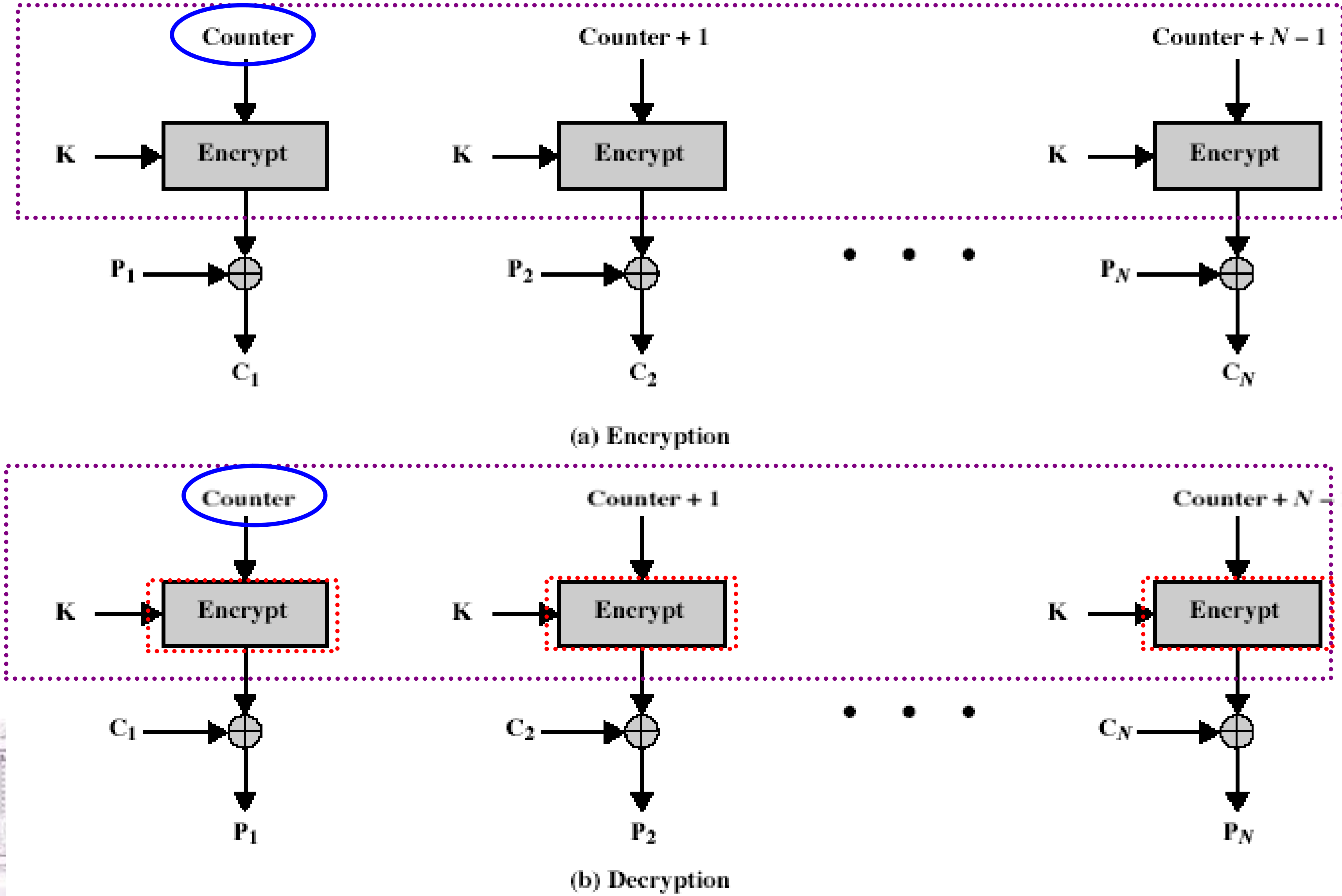
$$C_i = P_i \text{ XOR } O_i$$

$$O_i = E_K(i)$$

- uses: high-speed network encryptions

Counter (CTR)

$C_i = P_i \text{ XOR } O_i$
$O_i = E_K(i)$



Example

$C = C_1 C_2 C_3 C_4 C_5 \ \ldots$

$P = P_1 P_2 P_3 P_4 P_5 \ \ldots$

$C_i = P_i \text{ XOR } O_i$
$O_i = E_K(i)$

Case1:

$C' = C_1 C'_2 C_3 C_4 C_5 C_6 \ \ldots$

$P' = P_1 P'_2 P_3 P_4 P_5 P_6 \ \ldots$

Case2:

$C' = C_1 C_2 C_2 C_3 C_4 C_5 C_6 \ \ldots$

$P' = P_1 P_2 P'_3 P'_4 P'_5 P'_6 P'_7 \ \ldots$

Case3:

$C' = C_1 C_2 C_4 C_5 C_6 \ \ldots$

$P' = P_1 P_2 P'_3 P'_4 P'_5 \ldots$

Q:

$C' = C_1 C'_2 C'_3 C'_4 C_5 C_6 C_7 C_8 \ \ldots$

$P' = P_1 P'_2 P'_3 P'_4 P_5 P_6 P_7 P_8 \ \ldots$

IV for CTR

- Require to be merely unique
- Why?

$C_i = P_i$ XOR O_i

$O_i = E_K(i)$

If IV and k are reused for P and P',

then

Key steam $O = O_1 O_2 O_3 O_4 O_5 O_6 O_7 \ldots$

C= O XOR P, C'= O XOR P'

so, C XOR C' = P XOR P'

Characteristic of CTR

- efficiency
  - can do parallel encryptions in h/w or s/w
  - can preprocess in advance of need, which can greatly enhances throughput
- random access to encrypted data blocks
- provable security (good as other modes)
- but must ensure never reuse key/counter values, otherwise could break (similar to OFB)

Five ... 38A

- Ele ...)
- Cip
- Cip
- Ou
- Co

Mode	Description	Typical Application
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 *j* 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 DES output.	• 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

How to use block cipher?

- Keying Your Cipher
  - Forget about embedding the key in your application. It just will not work.
  - from a *master key* (also known as a *shared secret)* by using a *key derivation algorithm* such as PKCS #5
  - the master key should be random for every session (that is, each time a new message is to be processed), so the derived cipher key and IV should be random as well

- **Choosing a Chaining Mode**
  - Choosing between CBC and CTR mode is fairly simple.
  - They are both relatively the same speed in most software implementations.
  - CTR is more flexible in that it can natively support any length plaintext without padding or ciphertext stealing.
  - CBC is a bit more established in existing standards, on the other hand.
  - A simple rule of thumb(常用规则) is, unless you have to use CBC mode, choose CTR instead.

Outline

- Modes of Operation
- Random Bit Generation
- Stream Ciphers
  - RC4
- Purpose of Symmetric Encryption

Random Bit Generation

- Explain the concepts of randomness and unpredictability with respect to random numbers.
- Understand the differences among true random number generators (TRNG), pseudo-random number generators (PRNG), and pseudo-random functions(PRF).
- Present an overview of requirements for pseudorandom number generators.
- Explain how to construct a pseudorandom number generator.

The Use of Random Numbers

- **A number of network security algorithms and protocols based on cryptography make use of random binary numbers.**
  - **Key distribution and reciprocal (mutual) authentication schemes (discussed in Chapters 14 and 15 later).**
  - **Session key generation**
  - **Generation of keys for the RSA public-key encryption algorithm (described in Chapter 9).**
  - **Generation of a bit stream for symmetric stream encryption.**

Requirements for a sequence of random numbers

- Randomness
  - Uniform distribution
  - Independence
- Unpredictability

TRNG = true random number generator
PRNG = pseudorandom number generator
PRF = pseudorandom function

TRNGs, PRNGs, and PRFs

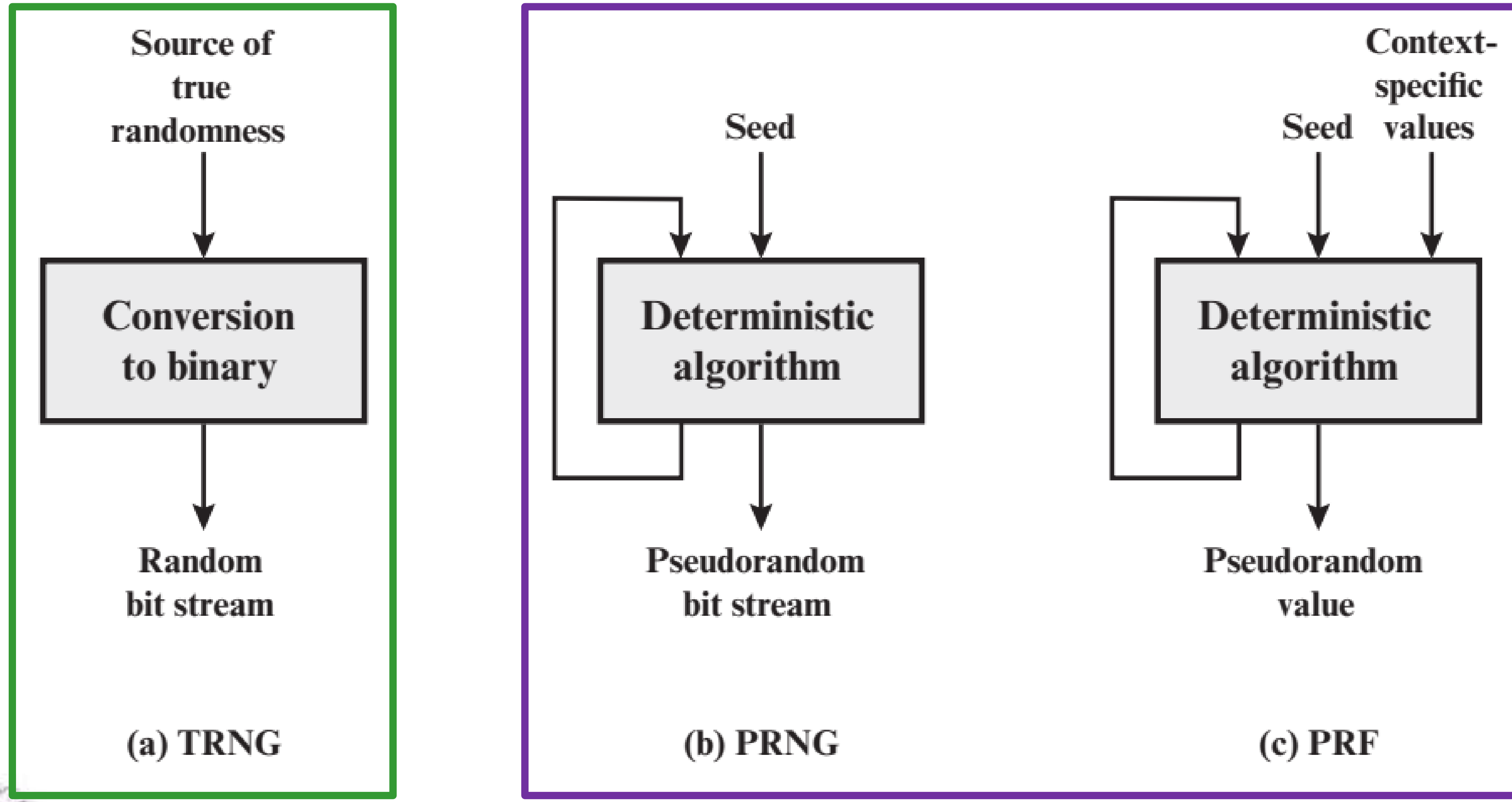


Figure 8.1 Random and Pseudorandom Number Generators

TRNG = true random number generator
PRNG = pseudorandom number generator
PRF = pseudorandom function

TRNGs, PRNGs, and PRFs

Table 8.5 Comparison of PRNGs and TRNGs

	Pseudorandom Number Generators	True Random Number Generators
Efficiency	Very efficient	Generally inefficient
Determinism	Deterministic	Nondeterministic
Periodicity	Periodic	Aperiodic

TRNG = true random number generator
PRNG = pseudorandom number generator
PRF = pseudorandom function

TRNGs, PRNGs, and PRFs

- **TRNG uses a nondeterministic source to produce randomness.**
  - **Sound/video input:**
  - **Disk drives:**
- **How to generate TRNG:**
  - **A hardware noise source produces a true random output.**
  - **digitized to produce true, or nondeterministic, source of bits.**
  - **Bit source then passes through a conditioning module to mitigate bias and maximize entropy.**

PRNG Requirements

- **Basic requirement is that an adversary who does not know the seed is unable to determine the pseudorandom string. (Secrecy of the output of a PRNG)**
  - **Randomness**
  - **Unpredictability**
    - **Forward unpredictability**
    - **Backward unpredictability**
  - **Seed Requirements**
    - **the seed must be unpredictable.**
    - **the seed itself must be a random or pseudo-random number.**

PRNG Requirements

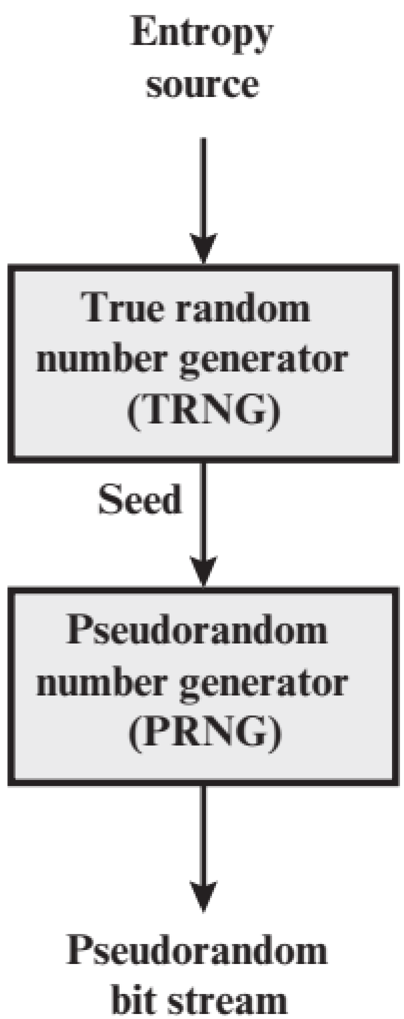


Figure 8.2 Generation of Seed Input to PRNG

PRNGs Algorithm Design

- **Purpose-built algorithms:**
  - **Designed specifically and solely for the purpose of generating pseudorandom bit streams.**
- **Algorithms based on existing cryptographic algorithms:**
  - **Cryptographic algorithms have the effect of randomizing input data.**
  - **Three categories of cryptographic algorithms used to create PRNGs:**
    - **Symmetric block ciphers**
    - **Asymmetric ciphers**
    - **Hash functions and message authentication codes**

Pseudorandom Number Generators

- **Linear Congruential Generators**

The sequence of random numbers $\{X_n\}$ is obtained via

$$X_{n+1} = (aX_n + c) \bmod m$$

- **Blum Blum Shub (BBS) Generator**

produces a sequence of bits B_i according to the following algorithm:

$$
\begin{aligned}
& X_0 = s^2 \bmod n \\
& \textbf{for } i = 1 \textbf{ to } \infty \\
& \quad X_i = (X_{i-1})^2 \bmod n \\
& \quad B_i = X_i \bmod 2
\end{aligned}
$$

Pseudorandom Number Generators

- **PRNGs using a block cipher:**
  - **PRNG Using Block Cipher Modes of Operation**

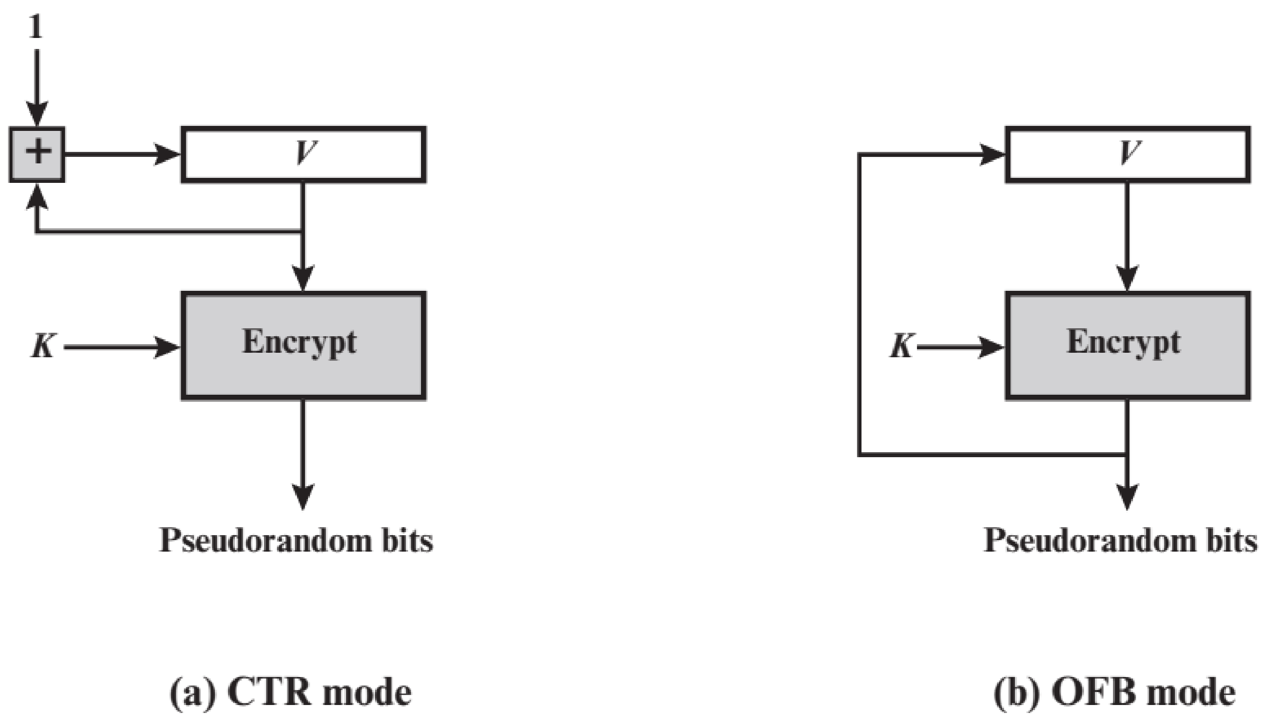


Figure 8.4 PRNG Mechanisms Based on Block Ciphers

Pseudorandom Number Generators

- **PRNGs using a block cipher:**
  - ANSI X9.17 PRNG

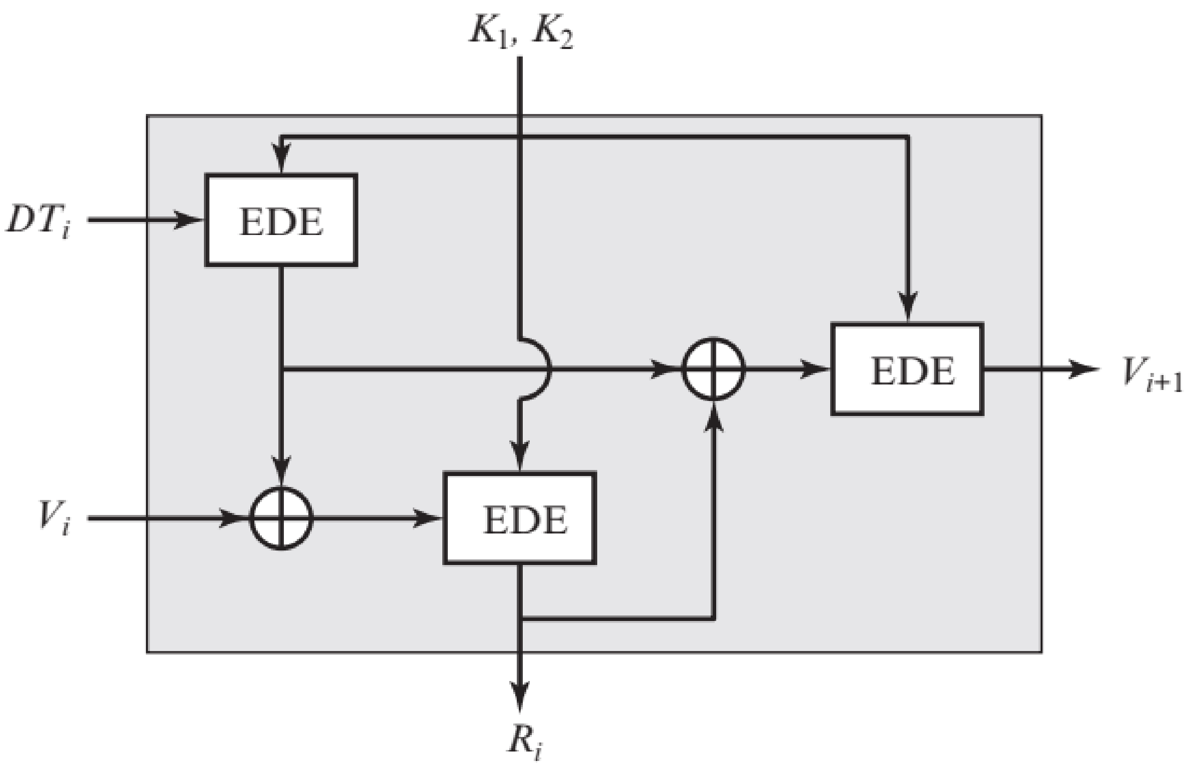


Figure 8.5 ANSI X9.17 Pseudorandom Number Generator

Pseudorandom Number Generators

- **PRNGs using a block cipher:**
  - **NIST CTR_DRBG(Counter mode–deterministic random bit generator)**

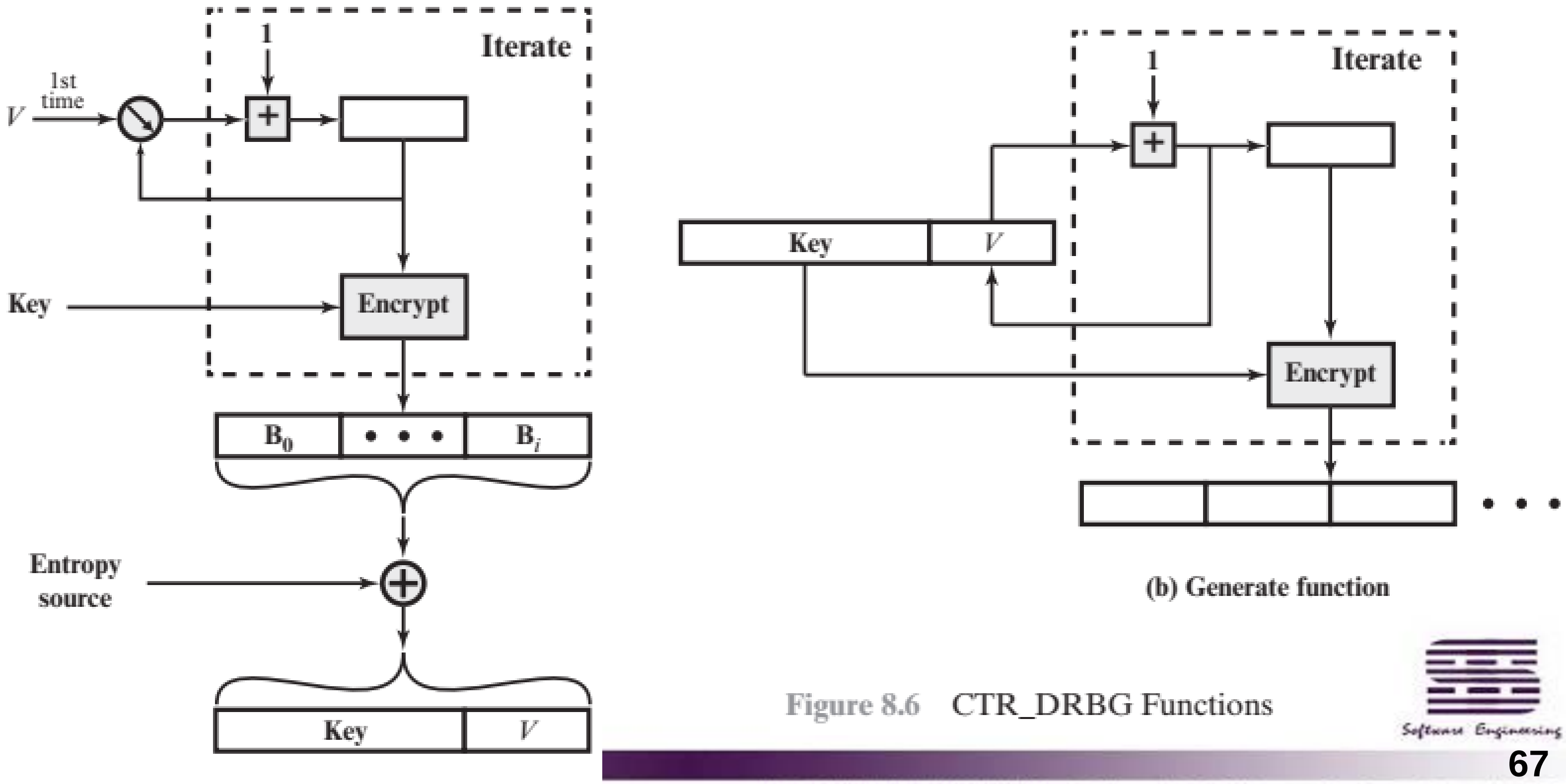


Figure 8.6 CTR_DRBG Functions

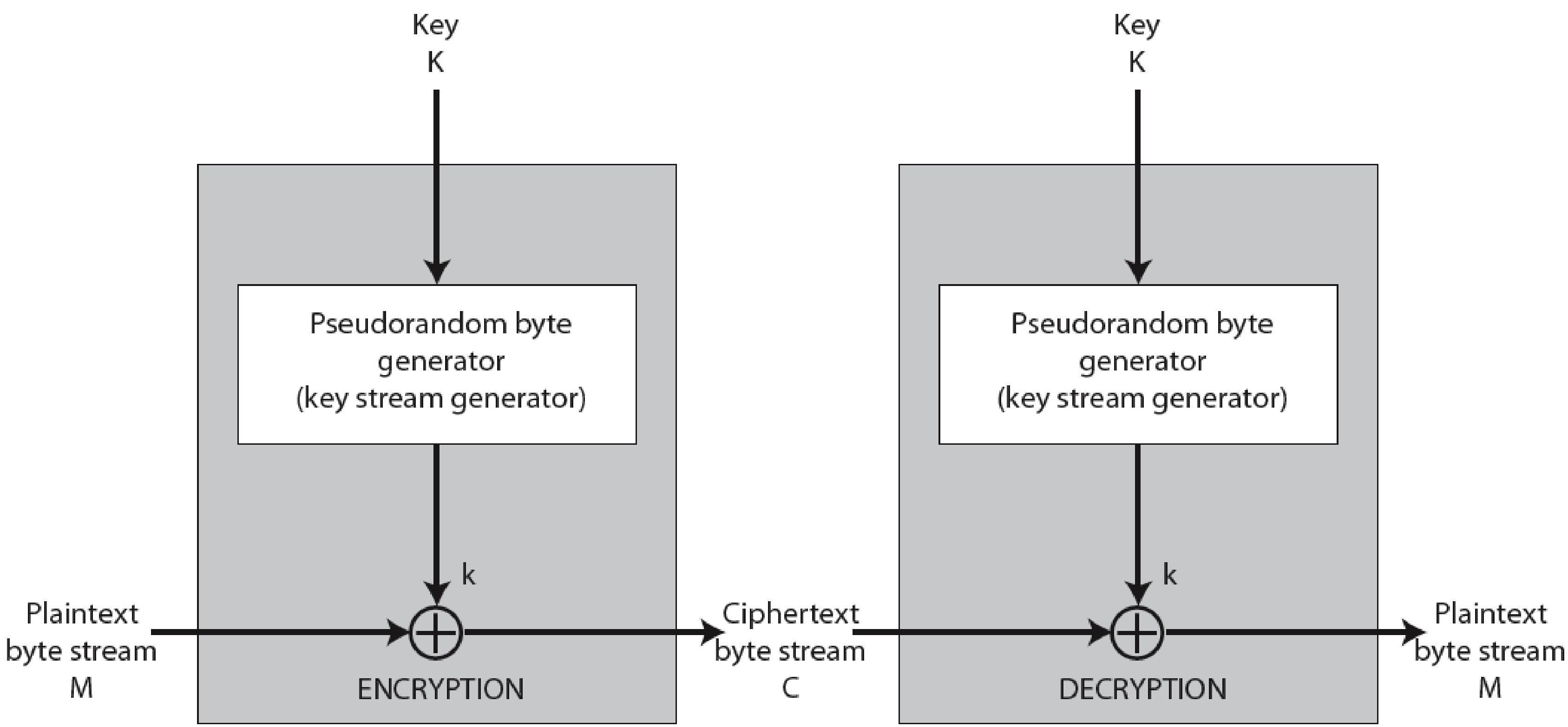
Outline

- Modes of Operation
- Random Bit Generation
- Stream Ciphers
  - RC4
- Purpose of Symmetric Encryption

Stream Ciphers

- **process message bit by bit (as a stream)**
- **have a pseudo random** keystream **combined (XOR) with plaintext bit by bit**
- **randomness of** stream key **completely destroys statistically properties in message**
  - $C_i = M_i \text{ XOR } StreamKey_i$
- **but must never reuse stream key**
  - **otherwise can recover messages**

Stream Cipher Structure



Stream Cipher Properties

- some design considerations of key stream are:
  - long period with no repetitions
  - statistically random
  - depends on large enough key
- properly designed, can be as secure as a block cipher with same size key
- but usually simpler & faster

- Block cipher: can use key repeatedly

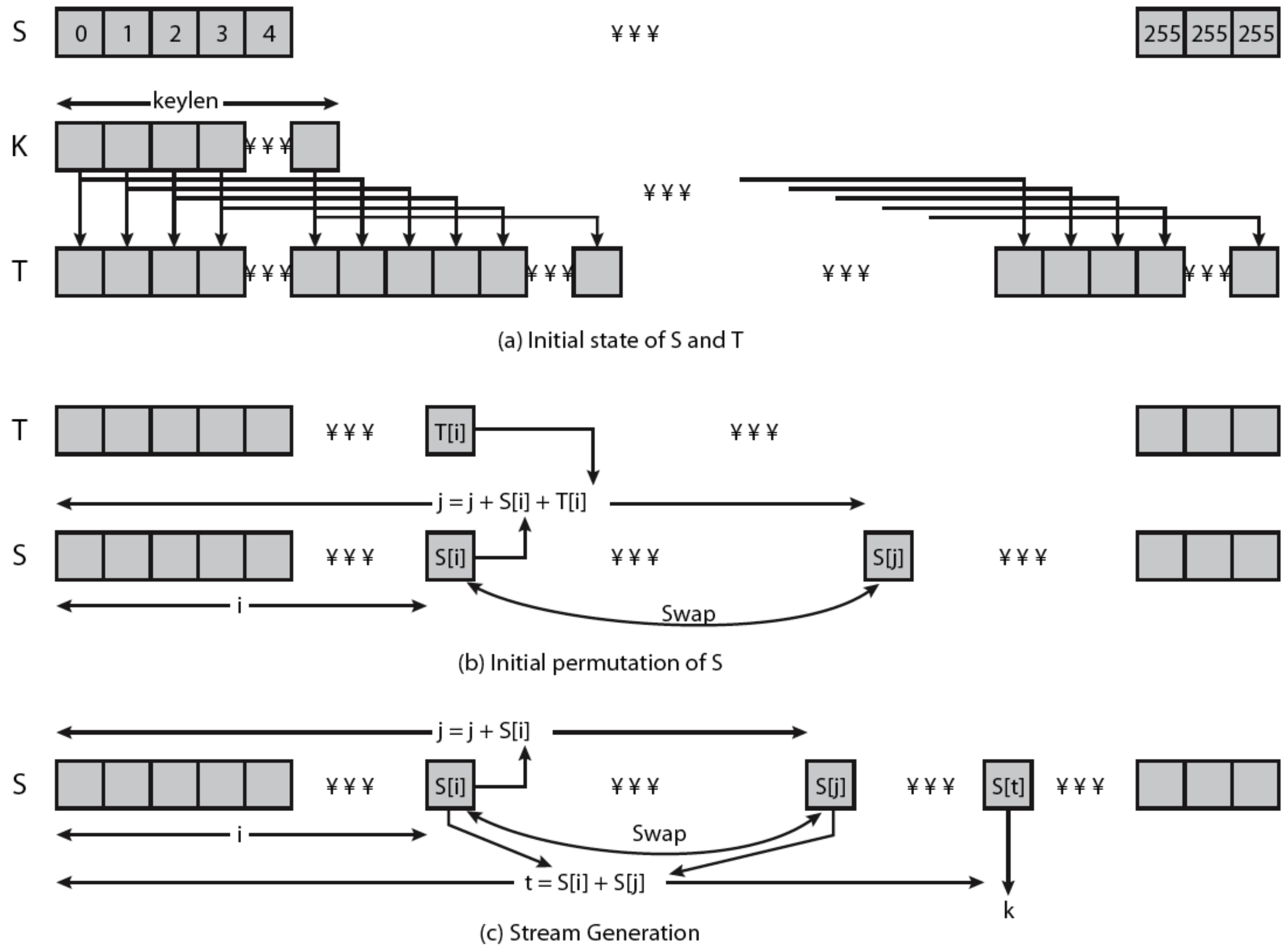
Table 6.2. Speed Comparisons of Symmetric Ciphers on a Pentium II

Cipher	Key Length	Speed (Mbps)
DES	56	9
3DES	168	3
RC2	variable	0.9
RC4	variable	45

RC4 *1987*

- a proprietary cipher owned by RSA DSI
- another Ron Rivest design, simple but effective
- variable key size, byte-oriented stream cipher
- widely used (web SSL/TLS, wireless WEP)
- key forms random permutation of all 8-bit values
- uses that permutation to scramble input info processed a byte at a time

RC4 Overview



(a) Initial state of S and T
(b) Initial permutation of S
(c) Stream Generation

```
/* Initialization */
for  i = 0 to 255 do
S[i] = i;
T[i] = K[i mod keylen];

/* Initial Permutation of S */
j = 0;
for  i = 0 to 255 do
  j = (j + S[i] + T[i]) mod 256;
  Swap (S[i], S[j]);

/* Stream Generation */
i, j = 0;
while (true)
  i = (i + 1) mod 256;
  j = (j + S[i]) mod 256;
  Swap (S[i], S[j]);
  t = (S[i] + S[j]) mod 256;
  k = S[t];
```

simpler & faster!

RC4 Security

- **claimed secure against known attacks**
  - **have some analyses, none practical**
- **result is very non-linear**
- **since RC4 is a stream cipher, must** never reuse a key
- **have a concern with WEP, but due to key handling rather than RC4 itself.**
- **More recently, [PAUL07] revealed a more fundamental vulnerability in the RC4 key scheduling algorithm that reduces the amount of effort to discover the key.**

RC4 Security

- **IETF issued RFC 7465 prohibiting theuse of RC4 in TLS (Prohibiting RC4 Cipher Suites, February 2015).**
- **NIST also prohibited the use of RC4 for government use (SP 800-52, Guidelines for the Selection, Configuration, and Use of Transport Layer Security(TLS) Implementations, September 2013).**

Outline

- **Modes of Operation**
- **Random Bit Generation**
- **Stream Ciphers**
  - **RC4**
- **Purpose of Symmetric Encryption**

Table 1.4 Relationship Between Security Services and Mechanisms

SERVICE \ MECHANISM	Encipherment	Digital signature	Access control	Data integrity	Authentication exchange	Traffic padding	Routing control	Notarization
Peer entity authentication	Y	Y			Y			
Data origin authentication	Y	Y						
Access control			Y					
Confidentiality	Y						Y	
Traffic flow confidentiality	Y					Y	Y	
Data integrity	Y	Y		Y				
Nonrepudiation		Y		Y				Y
Availability				Y	Y			

Purpose of Symmetric Encryption

- Data Confidentiality :
  - Secure communication: encrypt message
    - How to realize in real world? —— Next Lecture
  - Secure storage: encrypt file, software, disk, etc.
- Authentication: CMAC(CBC-MAC)
  - CBC mode can not provide authentication

Summary

- Modes of Operation
  - ECB, CBC, CFB, OFB, CTR
- stream ciphers
  - RC4
  - simpler & faster
- Purpose of Symmetric Encryption

Review Questions

7.1 What is triple encryption?

7.2 What is a meet-in-the-middle attack?

7.3 How many keys are used in triple encryption?

7.4 List and briefly define the block cipher modes of operation.

7.5 Why do some block cipher modes of operation only use encryption while others use both encryption and decryption?

Problems

7.4 With the ECB mode, if there is an error in a block of the transmitted ciphertext, only the corresponding plaintext block is affected. However, in the CBC mode, this error propagates. For example, an error in the transmitted C_1 (Figure 7.4) obviously corrupts P_1 and P_2.

a. Are any blocks beyond P_2 affected?

b. Suppose that there is a bit error in the source version of P_1. Through how many ciphertext blocks is this error propagated? What is the effect at the receiver?

7.8 If a bit error occurs in the transmission of a ciphertext character in 8-bit CFB mode, how far does the error propagate?

Thanks!