Secret-Key Encryption and Operation Modes

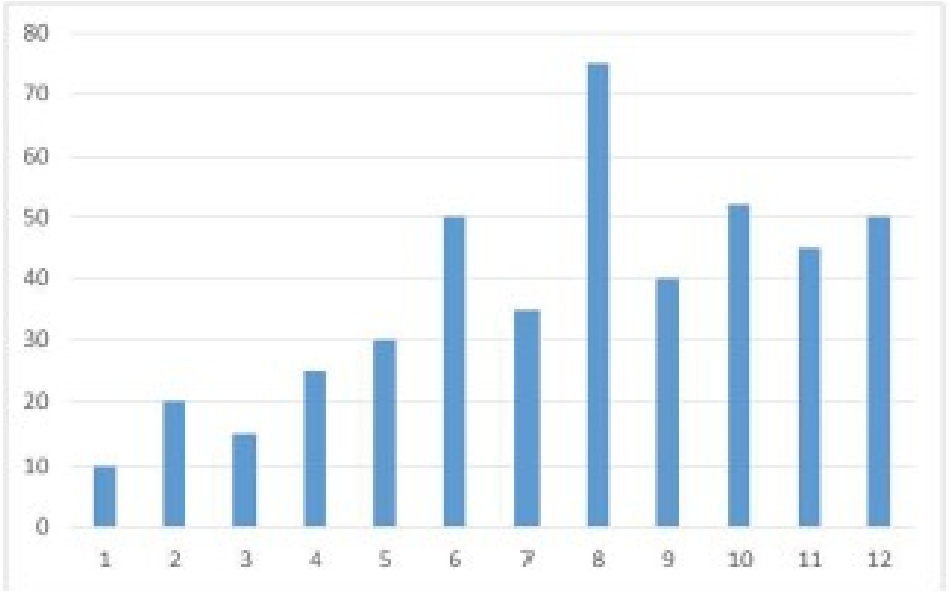
Data Encryption Standard (DES)

- DES is a block cipher - can only encrypt a block of data
- Block size for DES is 64 bits
- DES uses 56-bit keys although a 64-bit key is fed into the algorithm
- Theoretical attacks were identified. None was practical enough to cause major concerns.
- Triple DES can solve DES’s key size problem

Advanced Encryption Standard (AES)

- AES is a block cipher
- 128-bit block size.
- Three different key sizes: 128, 192, and 256 bits

Encryption Modes



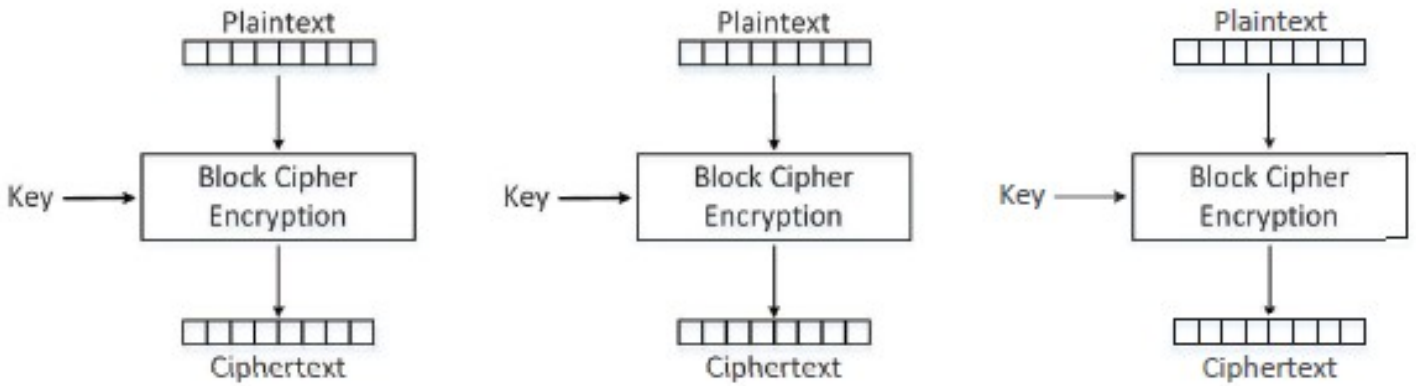
(a) The original image (`pic_original.bmp`)

(b) The encrypted image (`pic_encrypted.bmp`

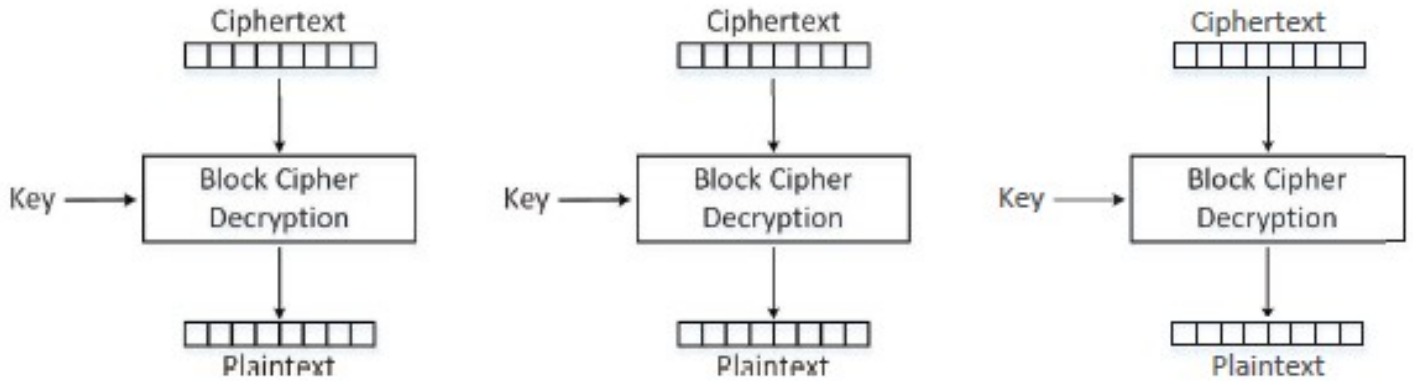
Encryption Modes

- Encryption mode or mode of operation refers to the many ways to make the input of an encryption algorithm different.
- Examples include:
  - Electronic Codebook (ECB)
  - Cipher Block Chaining (CBC)
  - Propagating CBC (PCBC)
  - Cipher Feedback (CFB)
  - Output Feedback (OFB)
  - Counter (CTR)

Electronic Codebook (ECB) Mode



(a) Electronic Codebook (ECB) mode encryption



(b) Electronic Codebook (ECB) mode decryption

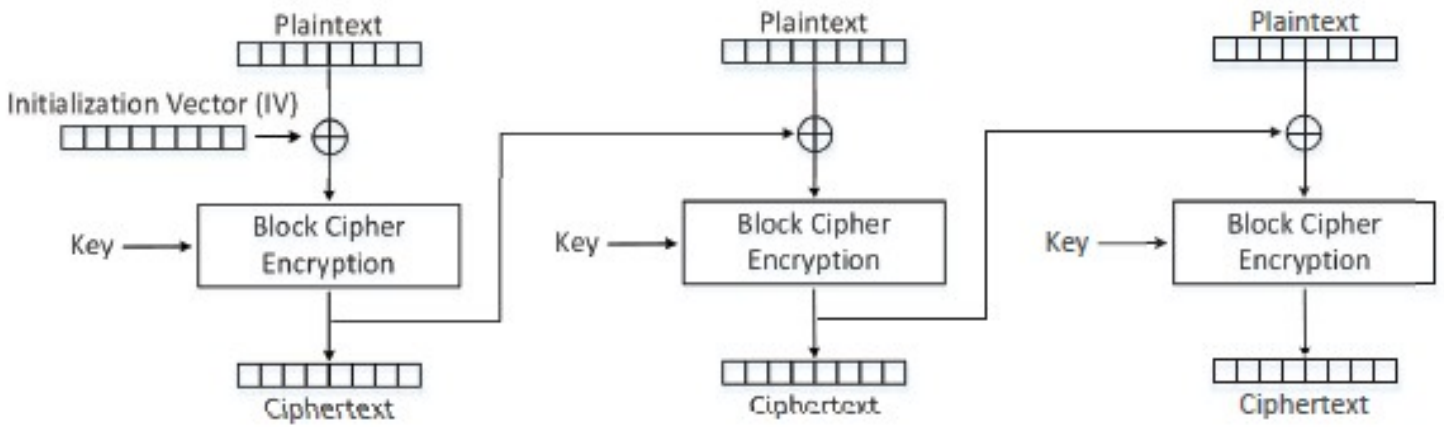
Electronic Codebook (ECB) Mode

- Using `openssl enc` command:

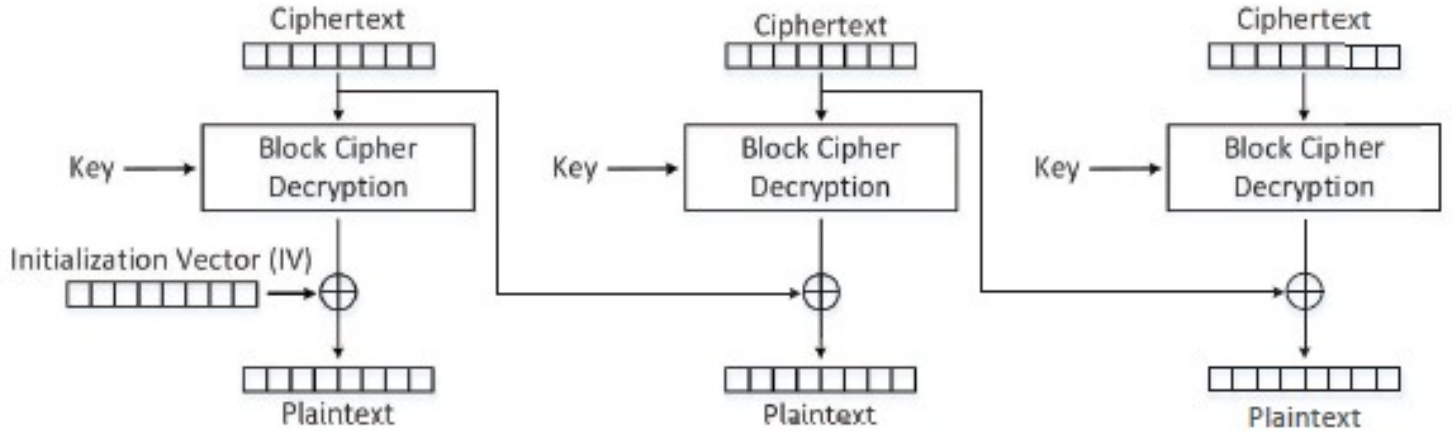
```
$ openssl enc -aes-128-ecb -e -in plain.txt -out cipher.txt \
          -K 00112233445566778899AABBCCDDEEFF
$ openssl enc -aes-128-ecb -d -in cipher.txt -out plain2.txt \
          -K 00112233445566778899AABBCCDDEEFF
```

- We use the 128-bit (key size) AES algorithm
- The **-aes-128-ecb** option specifies ECB mode
- The **-e** option indicates encryption
- The **-d** option indicate decryption
- The **-K** option is used to specify the encryption/decryption key

Cipher Block Chaining (CBC) Mode



(a) Cipher Block Chaining (CBC) mode encryption



(b) Cipher Block Chaining (CBC) mode decryption

- The main purpose of **IV** is to ensure that even if two plaintexts are identical, their ciphertexts are still different, because different IVs will be used.
- Decryption **can** be parallelized
- Encryption **cannot** be parallelized

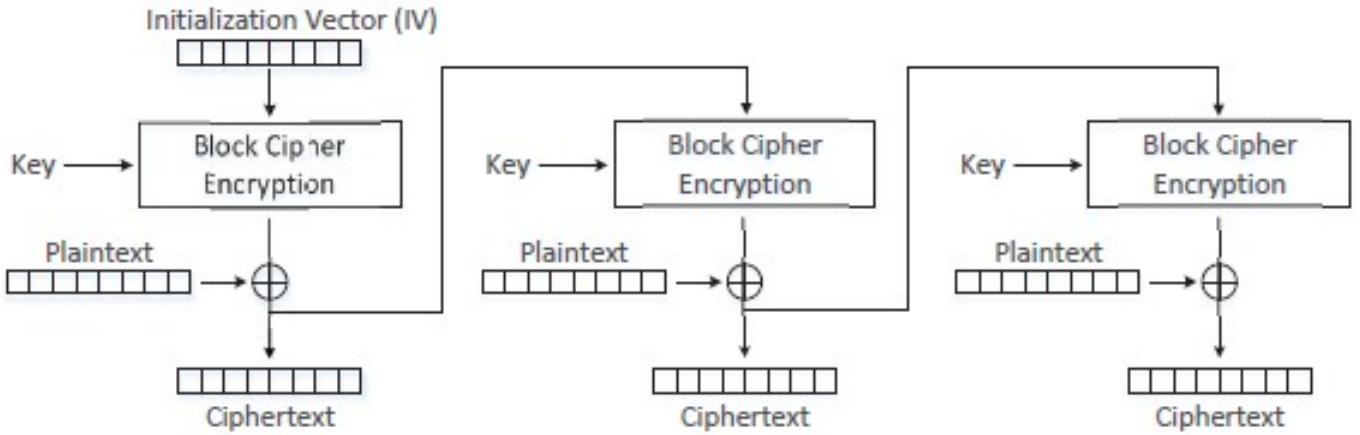
Cipher Block Chaining (CBC) Mode

- Using `openssl enc` command to encrypt the same plaintext, same key, **different IV**:

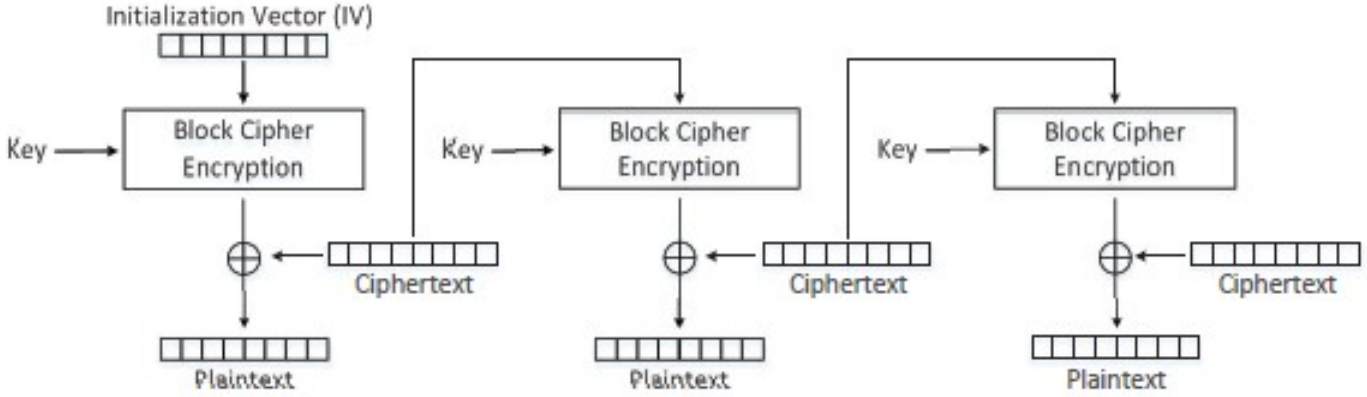
```
$ openssl enc -aes-128-cbc -e -in plain.txt -out cipher1.txt  \
              -K  00112233445566778899AABBCCDDEEFF \
              -iv 000102030405060708090a0b0c0d0e0f
$ openssl enc -aes-128-cbc -e -in plain.txt -out cipher2.txt  \
              -K  00112233445566778899AABBCCDDEEFF \
              -iv 000102030405060708090a0b0c0d0e0e
$ xxd -p cipher1.txt
52381c7726763ac132752bb29a32a68fc8dbcf20367fdfd03649b3a0d1744567
$ xxd -p cipher2.txt
50a9e3b81cc020d286d86fc7f1d8fb4268f9cd87c08126226c4626dbd4961d58
```

- We use the 128-bit (key size) AES algorithm
- The **-aes-128-cbc** option specifies CBC mode
- The **-e** option indicates encryption
- The **-iv** option is used to specify the Initialization Vector (IV)

Cipher Feedback (CFB) Mode



(a) Cipher Feedback (CFB) mode encryption



(b) Cipher Feedback (CFB) mode decryption

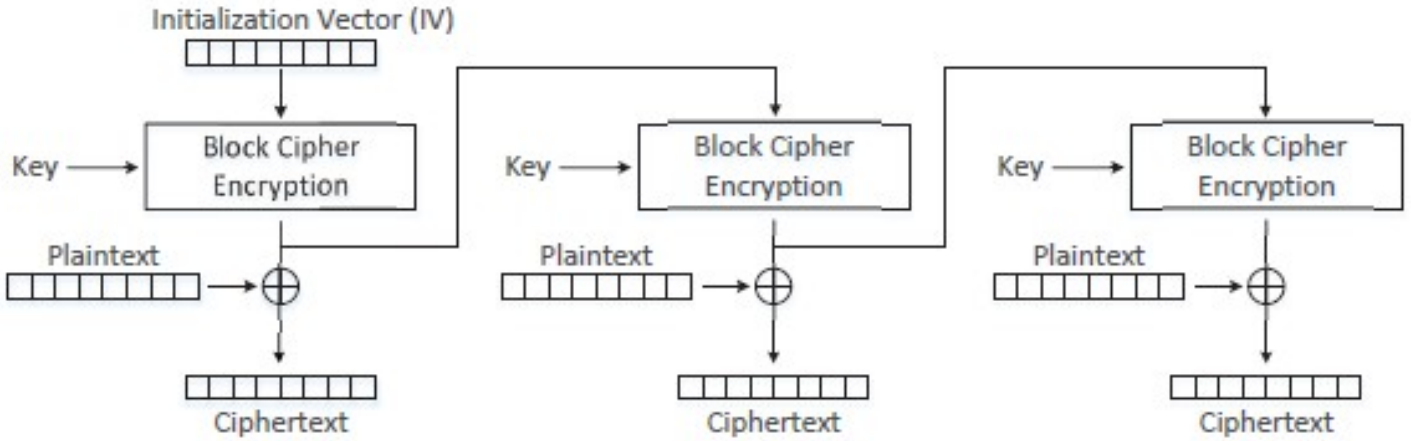
- A block cipher is turned into a stream cipher.
- Ideal for encrypting real-time data.
- Padding not required for the last block.
- decryption using the CFB mode can be parallelized, while encryption can only be conducted sequentially

Comparing encryption with CBC and CFB

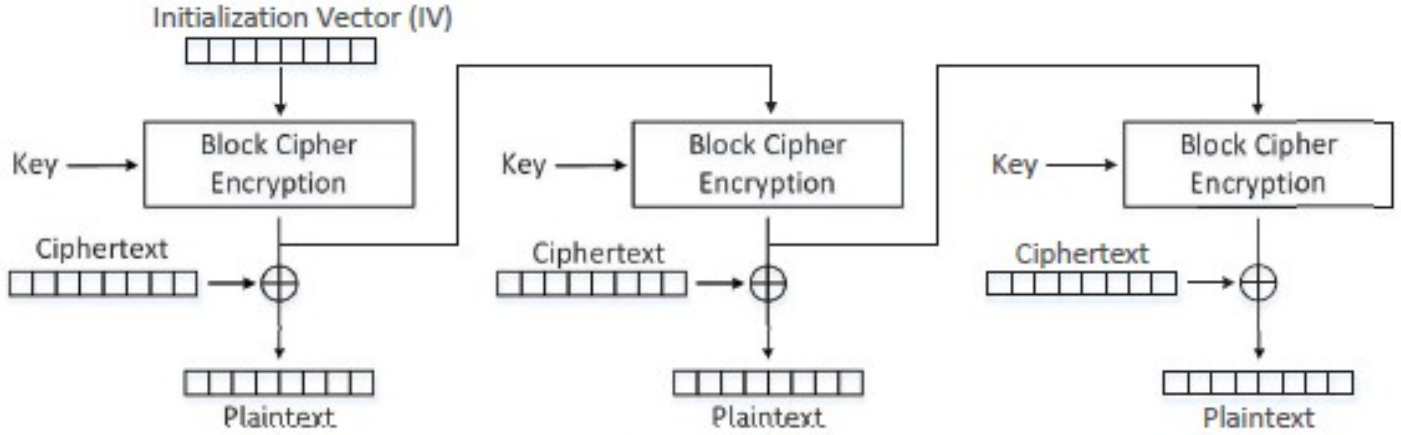
```
$ openssl enc -aes-128-cbc -e -in plain.txt -out cipher1.txt  \
              -K  00112233445566778899AABBCCDDEEFF \
              -iv 000102030405060708090a0b0c0d0e0f
$ openssl enc -aes-128-cfb -e -in plain.txt -out cipher2.txt  \
              -K  00112233445566778899AABBCCDDEEFF \
              -iv 000102030405060708090a0b0c0d0e0f
$ ls -l plain.txt cipher1.txt cipher2.txt
-rw-rw-r-- 1 seed seed 32 Jun 20 13:55 cipher1.txt
-rw-rw-r-- 1 seed seed 21 Jun 20 13:55 cipher2.txt
-rw-rw-r-- 1 seed seed 21 May 11 10:27 plain.txt
```

- Plaintext size is 21 bytes
- CBC mode: ciphertext is 32 bytes due padding
- CFB mode: ciphertext size is same as plaintext size (21 bytes)

Output Feedback (OFB) Mode



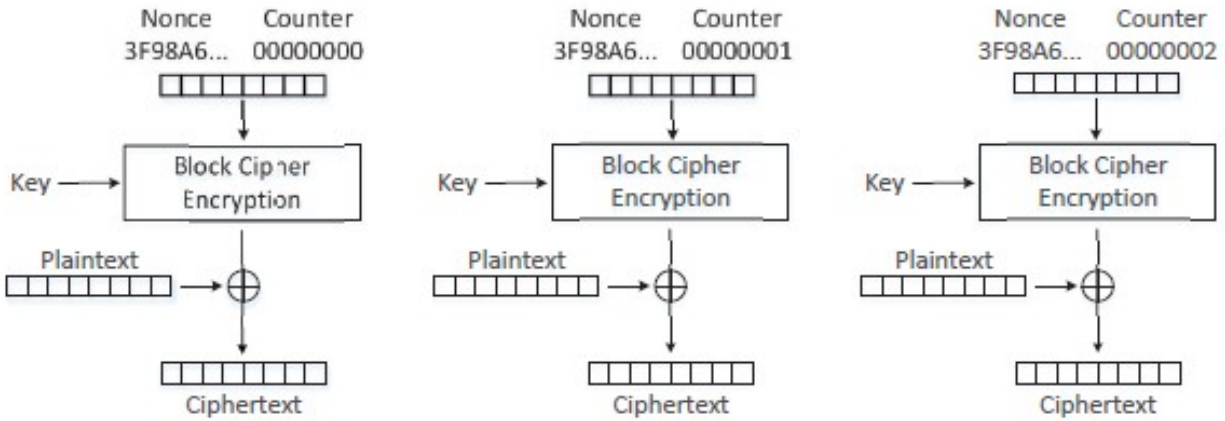
(a) Output Feedback (OFB) mode encryption



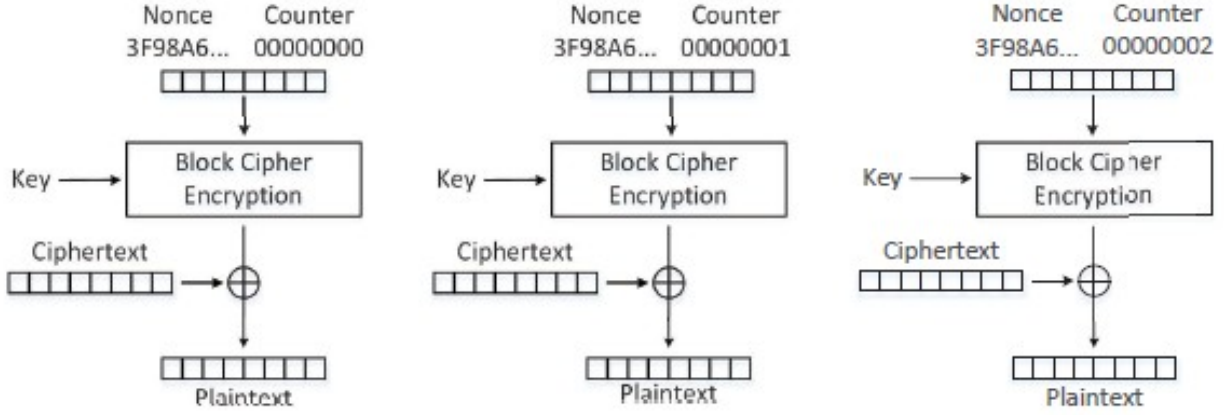
(b) Output Feedback (OFB) mode decryption

- Similar to CFB
  - Used as stream cipher
  - Does not need padding
  - Decryption can parallelized (see below)
- Encryption in the OFB mode can be parallelized
  - Block ciphering is made offline
  - XOR operation is parallelized

Counter (CTR) Mode



(a) Counter (CTR) mode encryption



(b) Counter (CTR) mode decryption

- It basically uses a counter to generate the key streams
- no key stream can be reused, hence the counter value for each block is prepended with a randomly generated value called *nonce*
- This nonce serves the same role as the IV does to the other encryption modes.
- both encryption and decryption can be parallelized
- the key stream in the CTR mode can be calculated in parallel during the encryption

Modes for Authenticated Encryption

- None of the Encryption modes discussed so far cannot be used to achieve message authentication
- A number of modes of operation have been designed to combine message authentication and encryption.
- Examples include
  - GCM (Galois/Counter Mode)
  - CCM (Counter with CBC-MAC)
  - OCB mode (Offset Codebook Mode)

Padding

- Block cipher encryption modes divide plaintext into blocks and the size of each block should match the cipher’s block size.
- No guarantee that the size of the last block matches the cipher’s block size.
- Last block of the plaintext needs **padding** i.e. before encryption, extra data needs to be added to the last block of the plaintext, so its size equals to the cipher’s block size.
- Padding schemes need to clearly mark where the padding starts, so decryption can remove the padded data.
- Commonly used padding scheme is PKCS#5

Padding Experiment

```
$ echo -n "123456789" > plain.txt
$ openssl enc -aes-128-cbc -e -in plain.txt -out cipher.bin  \
              -K  00112233445566778899AABBCCDDEEFF \
              -iv 0102030405060708090a0b0c0d0e0f
$ ls -ld cipher.bin
-rw-rw-r-- 1 seed seed 16 Jun 28 11:15 cipher.bin
$ openssl enc -aes-128-cbc -d -in cipher.bin -out plain2.txt \
              -K 00112233445566778889aabbccddeeff \
              -iv 0102030405060708
$ ls -ld plain2.txt
-rw-rw-r-- 1 seed seed  9 Jun 28 11:16 plain2.txt
```

- Plaintext size is 9 bytes.
- Size of ciphertext (cipher.bin) becomes 16 bytes

Initial Vector and Common Mistakes

- Initial vectors have the following requirements:
  - IV is supposed to be stored or transmitted in plaintext
  - IV should not repeat (uniqueness).
  - IV should not be predictable.

Experiment - IV should not be predictable

- Eve calculates the next IV

```
IV_bob:  4ae71336e44bf9bf79d2752e234818a5

# Encrypt Bob's vote
$ echo -n "John Smith......" > P1
$ openssl enc -aes-128-cbc -e -in P1 -out C1  \
              -K  00112233445566778899AABBCCDDEEFF \
              -iv 4ae71336e44bf9bf79d2752e234818a5

# Calculate IV_next from IV_bob
$ echo -n 4ae71336e44bf9bf79d2752e234818a5 | xxd -r -p > IV_bob
$ md5sum IV_bob
398d01fdf7934d1292c263d374778e1a

# Therefore, IV_next is 398d01fdf7934d1292c263d374778e1a
```

Experiment - IV should not be predictable

- Eve guesses that Bob voted for John Smith, so she creates *P1_guessed* and XOR it with IV_bob and IV_next, and finally constructs the name for a write-in candidate.

```
$ echo -n "John Smith......" > P1_guessed

# Convert the ascii string to hex string
$ xxd -p P1_guessed
4a6f686e20536d6974682e2e2e2e2e2e

# XOR P1_guessed with IV_bob
$ xor.py 4a6f686e20536d6974682e2e2e2e2e2e \
         4ae71336e44bf9bf79d2752e234818a5
00887b58c41894d60dba5b000d66368b

# XOR the above result with with IV_next
$ xor.py 00887b58c41894d60dba5b000d66368b \
         398d01fdf7934d1292c263d374778e1a
39057aa5338bd9c49f7838d37911b891

# Convert the above hex string to binary and save to P2
$ echo -n "39057aa5338bd9c49f7838d37911b891" | xxd -r -p > P2
```

Experiment - IV should not be predictable

- Eve gives her write-in candidate's name (stored in P2) to the voting machine, which encrypts the name using IV_next as the IV. The result is stored in C2.
- If C1 (Bob's encrypted vote) == C2, then Eve knows for sure that Bob has voted for "John Smith".

```
$ openssl enc -aes-128-cbc -e -in P2 -out C2  \
              -K  00112233445566778899AABBCCDDEEFF \
              -iv 398d01fdf7934d1292c263d374778e1a

# Compare C1 and C2
$ xxd -p C1
7380ee1c0f9eb7dae28c1ba6a1a74310114288f771139da8ec99dfb0036e38ce
$ xxd -p C2
7380ee1c0f9eb7dae28c1ba6a1a74310114288f771139da8ec99dfb0036e38ce
```