DES Documentation

Data Encryption Standard

documentation by Molly Taylor
August 2019

Contents

Chapter 1

Password-based key derivation function (PBKDF)

You typically begin with only a password. From the password, you must derive the key and IV, which are each 64-bits.

password: A string provided by the user.

salt: 64-bits which are used in the PBKDF. Unless it's provided by the user, it's randomly generated.

key: 64-bits from which the subkeys are derived. Unless specified, it's created by the PBKDF.

IV (initialization vector): 64-bits. Used in CBC mode. Unless specified, it's created by the PBKDF.

1.1 PBKDF1

1. Generate random salt
2. Concatenate salt to the end of password
3. Run MD5 on concatenated password

MD5's checksum is 128-bits. The first 64-bits are the key. The next 64-bits are the IV.

Chapter 2

Cipher

2.1 Pseudocode

Algorithm 1 Cipher

function CIPHER($block$, key)
    $subkeys[16]$ = key schedule(key)
    $block$ = IP($block$) ▷ initial permutation (IP)
    split into $left$ and $right$ halves
    for 16 rounds, **do**
        $left_{i+1} = right_i$
        $right_{i+1} = left_i \oplus$ Feistel($right_i, subkey[i]$)
    $block$ = concatenate halves
    $block = \text{IP}^{-1}(block)$ ▷ final permutation (IP^{-1})
    return $block$

Algorithm 2 Feistel function, also sometimes called the F-function

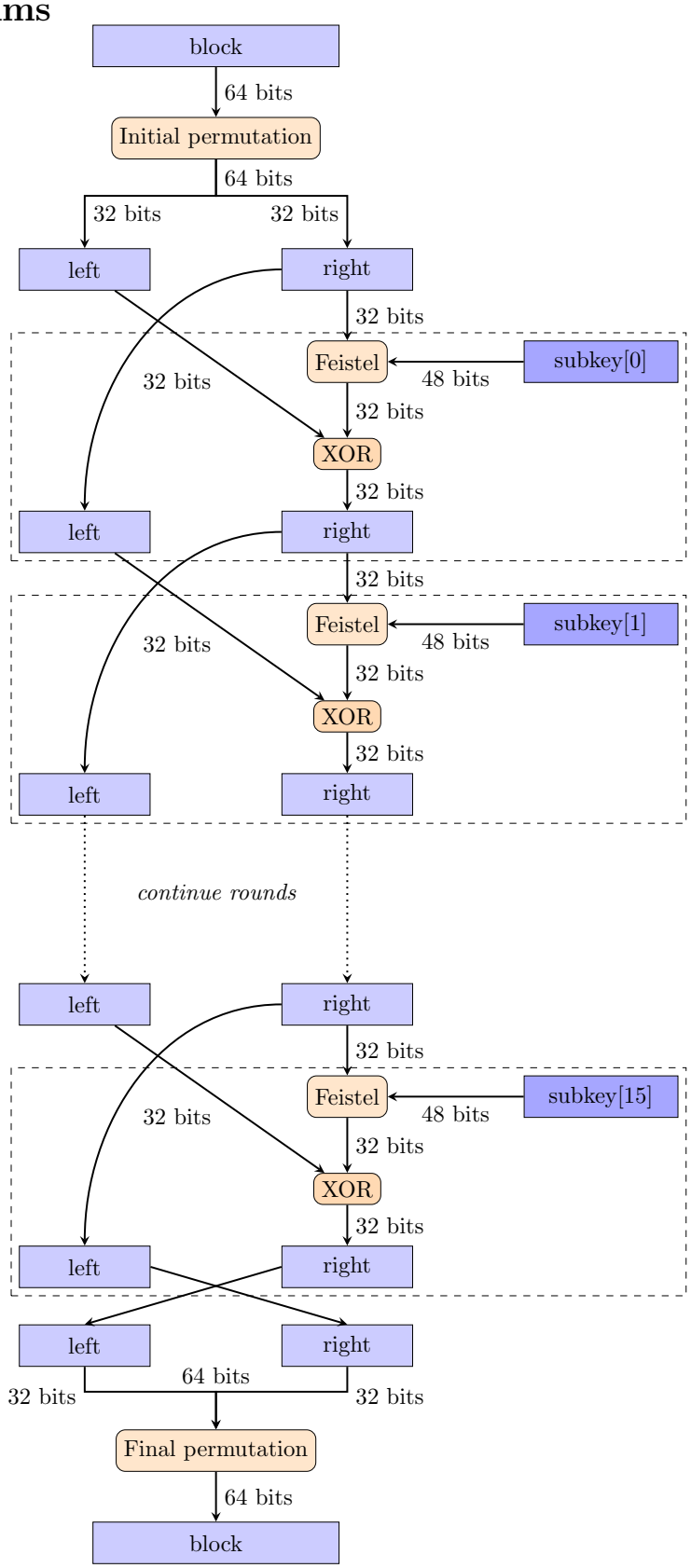
function FEISTEL($block$, $subkey$)
    $block$ = expansion($block$) ▷ expansion permutation (E)
    $block = block \oplus subkey$
    $block$ = s-boxes($block$) ▷ substitution boxes (s-boxes)
    $block$ = p-box($block$) ▷ p-box permutation (P)
    return $block$

Algorithm 3 Key schedule

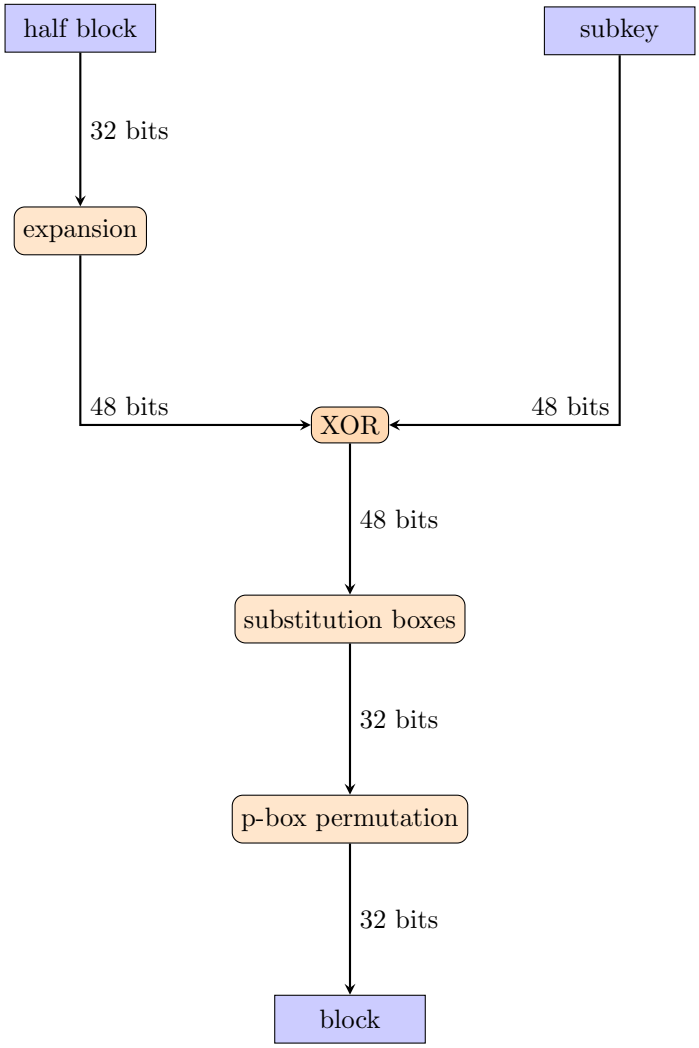
function KEY SCHEDULE(key)
    key = PC-1(key) ▷ permuted choice 1 (PC-1)
    split into $left$ and $right$ halves
    for 16 rounds, **do**
        $left$ = left rotate($left, rotation[i]$) ▷ rotation table
        $right$ = left rotate($right, rotation[i]$)
        $block$ = concatenate halves
        $subkey[i]$ = PC-2($block$) ▷ permuted choice 2 (PC-2)
    return $subkeys[16]$

2.2 Diagrams

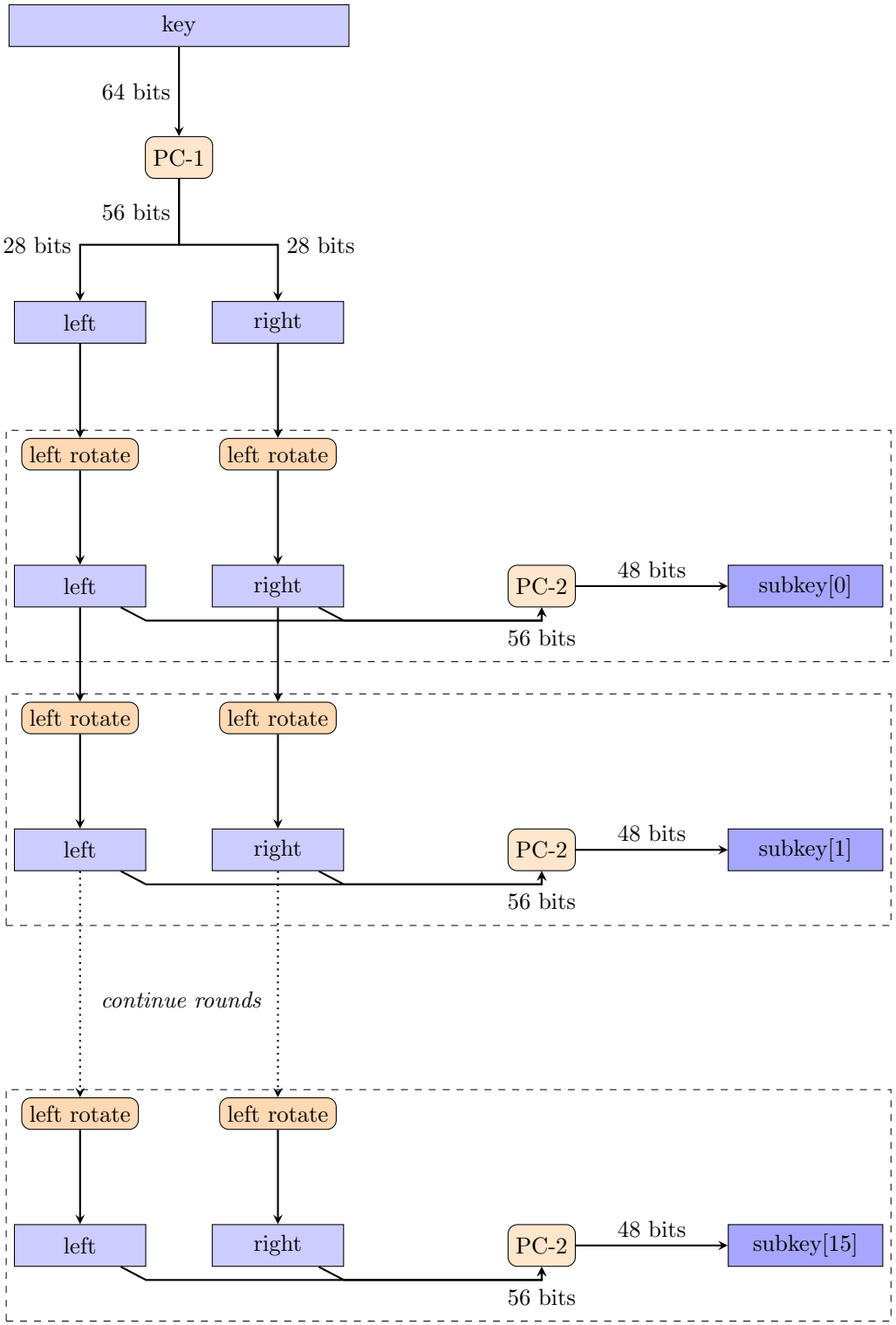
2.2.1 Cipher



2.2.2 Feistel


half block
subkey
32 bits
expansion
48 bits
XOR
48 bits
48 bits
substitution boxes
32 bits
p-box permutation
32 bits
block

2.2.3 Key schedule



Chapter 3

Permutations

Permutations involve reordering bits in accordance with a table. For example, if the first number of the table is 10, then the first bit of the output will be the 10th bit of the input.

permutation	input	output
initial permutation (IP)	64 bits	64 bits
final permutation (IP^{-1})	64 bits	64 bits
expansion permutation (E)	32 bits	48 bits
p-box permutation (P)	32 bits	32 bits
permuted choice 1 (PC-1)	64 bits	56 bits
permuted choice 2 (PC-2)	56 bits	48 bits

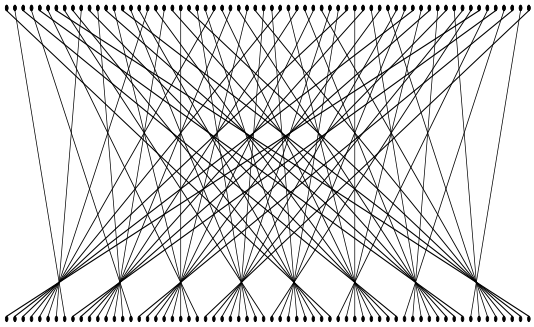


Figure 1: Initial permutation (IP)

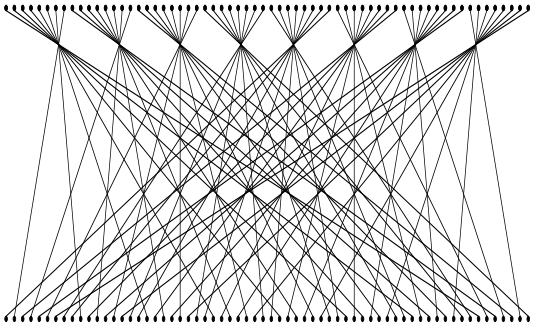


Figure 2: Final permutation (IP^{-1})

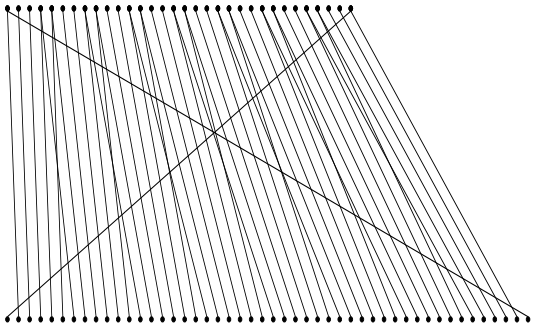


Figure 3: Expansion permutation (E)

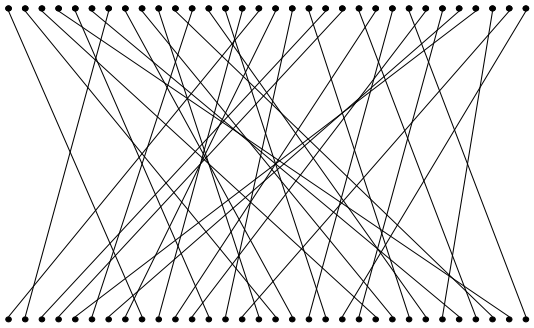


Figure 4: P-box permutation (P)

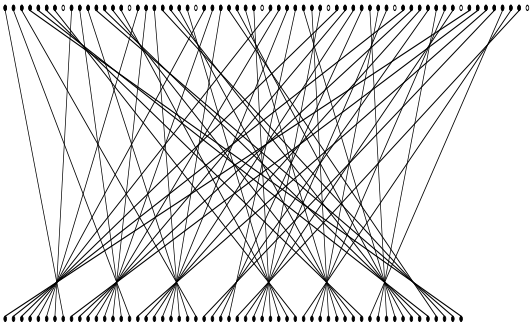


Figure 5: Permuted choice 1 (PC-1)

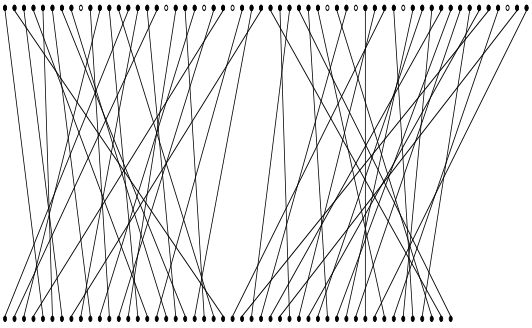


Figure 6: Permuted choice 2 (PC-2)

Chapter 4

Substitution boxes (s-boxes)

The 48-bit block is divided into 8 pieces of 6-bits each. Each 6-bit input is replaced with 4-bit output, according to the lookup table.

- the **nth of the sextet** refers to the **box**
- the first and last bit of the sextet are combined into a **2-bit value** which refers to the **row**
- the inner 4 bits of the sextet become a **4-bit value** which refers to the **column**

Algorithm 4 substitution boxes (s-boxes)

```
1: function S-BOXES(input)
2:     for 8 rounds, do
3:         sextet = input[i]                          ▷ input clumped by 6 bits
4:         outer = first and last bit of sextet
5:         inner = middle 4 bits of sextet
6:         quartet = table[i][outer][inner]           ▷ s-box lookup table
7:         output[i] = quartet                        ▷ output clumped by 4 bits
8:     return output
```

4.1 Example

Input (48-bits): `011000 010001 011110 111010 100001 100110 010100 100111`

sextet	s-box lookup
011000_2	sbox[0][00_2][1100_2] = 0101_2
010001_2	sbox[1][01_2][1000_2] = 1100_2
011110_2	sbox[2][00_2][1111_2] = 1000_2
111010_2	sbox[3][10_2][1101_2] = 0010_2
100001_2	sbox[4][11_2][0000_2] = 1011_2
100110_2	sbox[5][10_2][0011_2] = 0101_2
010100_2	sbox[6][00_2][1010_2] = 1001_2
100111_2	sbox[7][11_2][0011_2] = 0111_2

Output (32-bits): `0101 1100 1000 0010 1011 0101 1001 0111`

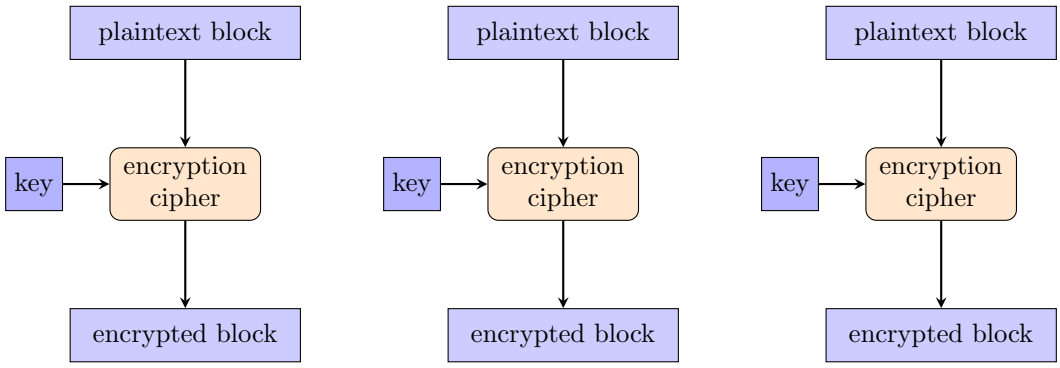
Chapter 5

Modes of operation

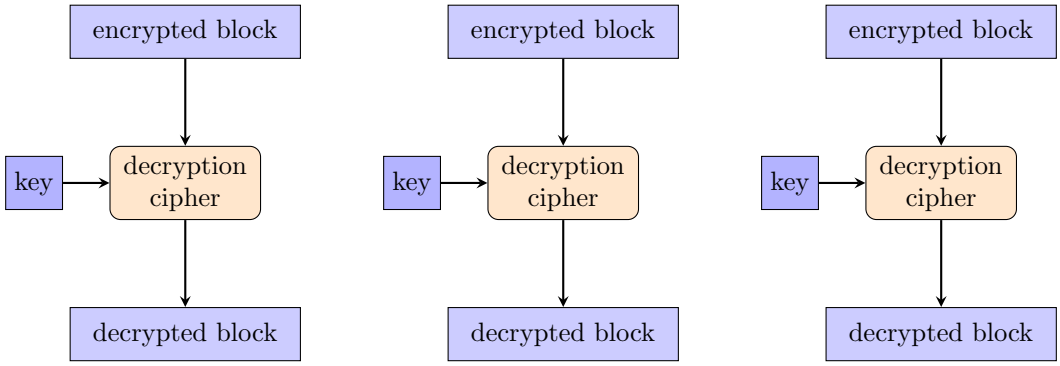
5.1 Electronic Codebook (ECB)

ECB is "plain." It's simply encrypted and decrypted block by block.

5.1.1 Encryption



5.1.2 Decryption

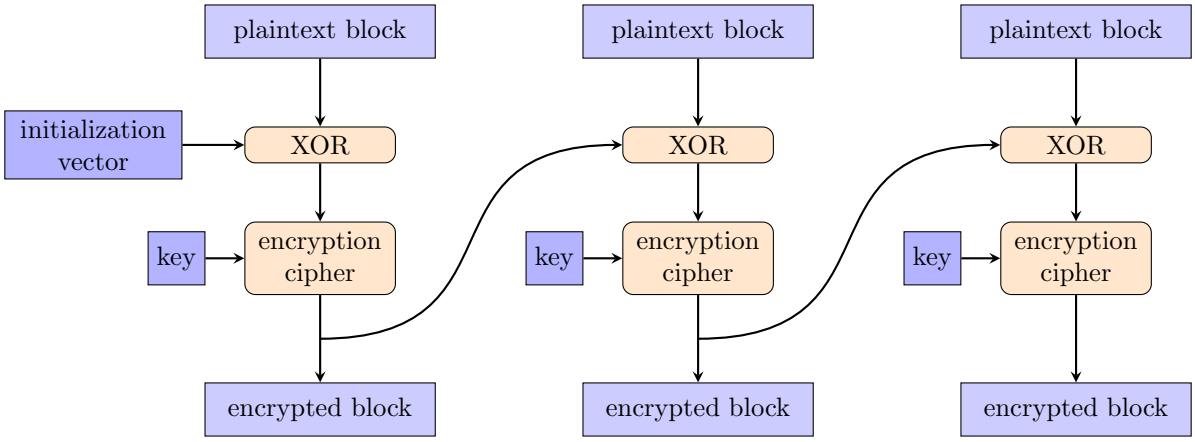


5.2 Cipher Block Chaining (CBC)

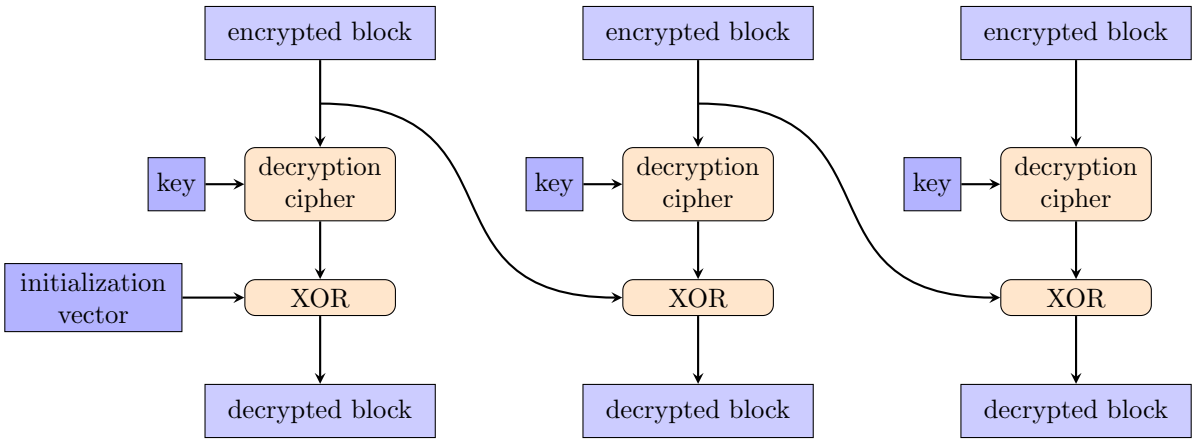
CBC has each block XORed with the previous block. The blocks are all linked together, like a chain. For the very first block, it's XORed with the IV (initialization vector).

- In encryption mode, XOR with the plaintext block (aka before encryption)
- In decryption mode, XOR with the decrypted block (aka after decryption)

5.2.1 Encryption



5.2.2 Decryption



Chapter 6

Lookup Tables

6.1 S-boxes

```
sboxes[8][4][16] = {
  {
    {0xe, 0x4, 0xd, 0x1, 0x2, 0xf, 0xb, 0x8, 0x3, 0xa, 0x6, 0xc, 0x5, 0x9, 0x0, 0x7},
    {0x0, 0xf, 0x7, 0x4, 0xe, 0x2, 0xd, 0x1, 0xa, 0x6, 0xc, 0xb, 0x9, 0x5, 0x3, 0x8},
    {0x4, 0x1, 0xe, 0x8, 0xd, 0x6, 0x2, 0xb, 0xf, 0xc, 0x9, 0x7, 0x3, 0xa, 0x5, 0x0},
    {0xf, 0xc, 0x8, 0x2, 0x4, 0x9, 0x1, 0x7, 0x5, 0xb, 0x3, 0xe, 0xa, 0x0, 0x6, 0xd},
  },
  {
    {0xf, 0x1, 0x8, 0xe, 0x6, 0xb, 0x3, 0x4, 0x9, 0x7, 0x2, 0xd, 0xc, 0x0, 0x5, 0xa},
    {0x3, 0xd, 0x4, 0x7, 0xf, 0x2, 0x8, 0xe, 0xc, 0x0, 0x1, 0xa, 0x6, 0x9, 0xb, 0x5},
    {0x0, 0xe, 0x7, 0xb, 0xa, 0x4, 0xd, 0x1, 0x5, 0x8, 0xc, 0x6, 0x9, 0x3, 0x2, 0xf},
    {0xd, 0x8, 0xa, 0x1, 0x3, 0xf, 0x4, 0x2, 0xb, 0x6, 0x7, 0xc, 0x0, 0x5, 0xe, 0x9},
  },
  {
    {0xa, 0x0, 0x9, 0xe, 0x6, 0x3, 0xf, 0x5, 0x1, 0xd, 0xc, 0x7, 0xb, 0x4, 0x2, 0x8},
    {0xd, 0x7, 0x0, 0x9, 0x3, 0x4, 0x6, 0xa, 0x2, 0x8, 0x5, 0xe, 0xc, 0xb, 0xf, 0x1},
    {0xd, 0x6, 0x4, 0x9, 0x8, 0xf, 0x3, 0x0, 0xb, 0x1, 0x2, 0xc, 0x5, 0xa, 0xe, 0x7},
    {0x1, 0xa, 0xd, 0x0, 0x6, 0x9, 0x8, 0x7, 0x4, 0xf, 0xe, 0x3, 0xb, 0x5, 0x2, 0xc},
  },
  {
    {0x7, 0xd, 0xe, 0x3, 0x0, 0x6, 0x9, 0xa, 0x1, 0x2, 0x8, 0x5, 0xb, 0xc, 0x4, 0xf},
    {0xd, 0x8, 0xb, 0x5, 0x6, 0xf, 0x0, 0x3, 0x4, 0x7, 0x2, 0xc, 0x1, 0xa, 0xe, 0x9},
    {0xa, 0x6, 0x9, 0x0, 0xc, 0xb, 0x7, 0xd, 0xf, 0x1, 0x3, 0xe, 0x5, 0x2, 0x8, 0x4},
    {0x3, 0xf, 0x0, 0x6, 0xa, 0x1, 0xd, 0x8, 0x9, 0x4, 0x5, 0xb, 0xc, 0x7, 0x2, 0xe},
  },
  {
    {0x2, 0xc, 0x4, 0x1, 0x7, 0xa, 0xb, 0x6, 0x8, 0x5, 0x3, 0xf, 0xd, 0x0, 0xe, 0x9},
    {0xe, 0xb, 0x2, 0xc, 0x4, 0x7, 0xd, 0x1, 0x5, 0x0, 0xf, 0xa, 0x3, 0x9, 0x8, 0x6},
    {0x4, 0x2, 0x1, 0xb, 0xa, 0xd, 0x7, 0x8, 0xf, 0x9, 0xc, 0x5, 0x6, 0x3, 0x0, 0xe},
    {0xb, 0x8, 0xc, 0x7, 0x1, 0xe, 0x2, 0xd, 0x6, 0xf, 0x0, 0x9, 0xa, 0x4, 0x5, 0x3},
  },
  {
    {0xc, 0x1, 0xa, 0xf, 0x9, 0x2, 0x6, 0x8, 0x0, 0xd, 0x3, 0x4, 0xe, 0x7, 0x5, 0xb},
    {0xa, 0xf, 0x4, 0x2, 0x7, 0xc, 0x9, 0x5, 0x6, 0x1, 0xd, 0xe, 0x0, 0xb, 0x3, 0x8},
    {0x9, 0xe, 0xf, 0x5, 0x2, 0x8, 0xc, 0x3, 0x7, 0x0, 0x4, 0xa, 0x1, 0xd, 0xb, 0x6},
    {0x4, 0x3, 0x2, 0xc, 0x9, 0x5, 0xf, 0xa, 0xb, 0xe, 0x1, 0x7, 0x6, 0x0, 0x8, 0xd},
  },
  {
    {0x4, 0xb, 0x2, 0xe, 0xf, 0x0, 0x8, 0xd, 0x3, 0xc, 0x9, 0x7, 0x5, 0xa, 0x6, 0x1},
    {0xd, 0x0, 0xb, 0x7, 0x4, 0x9, 0x1, 0xa, 0xe, 0x3, 0x5, 0xc, 0x2, 0xf, 0x8, 0x6},
    {0x1, 0x4, 0xb, 0xd, 0xc, 0x3, 0x7, 0xe, 0xa, 0xf, 0x6, 0x8, 0x0, 0x5, 0x9, 0x2},
    {0x6, 0xb, 0xd, 0x8, 0x1, 0x4, 0xa, 0x7, 0x9, 0x5, 0x0, 0xf, 0xe, 0x2, 0x3, 0xc},
  },
  {
    {0xd, 0x2, 0x8, 0x4, 0x6, 0xf, 0xb, 0x1, 0xa, 0x9, 0x3, 0xe, 0x5, 0x0, 0xc, 0x7},
    {0x1, 0xf, 0xd, 0x8, 0xa, 0x3, 0x7, 0x4, 0xc, 0x5, 0x6, 0xb, 0x0, 0xe, 0x9, 0x2},
    {0x7, 0xb, 0x4, 0x1, 0x9, 0xc, 0xe, 0x2, 0x0, 0x6, 0xa, 0xd, 0xf, 0x3, 0x5, 0x8},
    {0x2, 0x1, 0xe, 0x7, 0x4, 0xa, 0x8, 0xd, 0xf, 0xc, 0x9, 0x0, 0x3, 0x5, 0x6, 0xb},
  },
}
```

6.2 Permutations

Keep in mind that these permutation tables are not 0-indexed. The first bit is 1, not 0.

Table 6.1: Cipher permutations

```
initial_permutation[64] = {
    58, 50, 42, 34, 26, 18, 10, 2,
    60, 52, 44, 36, 28, 20, 12, 4,
    62, 54, 46, 38, 30, 22, 14, 6,
    64, 56, 48, 40, 32, 24, 16, 8,
    57, 49, 41, 33, 25, 17,  9, 1,
    59, 51, 43, 35, 27, 19, 11, 3,
    61, 53, 45, 37, 29, 21, 13, 5,
    63, 55, 47, 39, 31, 23, 15, 7
}
```

```
final_permutation[64] = {
    40, 8, 48, 16, 56, 24, 64, 32,
    39, 7, 47, 15, 55, 23, 63, 31,
    38, 6, 46, 14, 54, 22, 62, 30,
    37, 5, 45, 13, 53, 21, 61, 29,
    36, 4, 44, 12, 52, 20, 60, 28,
    35, 3, 43, 11, 51, 19, 59, 27,
    34, 2, 42, 10, 50, 18, 58, 26,
    33, 1, 41,  9, 49, 17, 57, 25
}
```

Table 6.2: Feistel permutations

```
expansion_permutation[48] = {
    32,  1,  2,  3,  4,  5,  4,  5,
     6,  7,  8,  9,  8,  9, 10, 11,
    12, 13, 12, 13, 14, 15, 16, 17,
    16, 17, 18, 19, 20, 21, 20, 21,
    22, 23, 24, 25, 24, 25, 26, 27,
    28, 29, 28, 29, 30, 31, 32,  1
}
```

```
pbox_permutation[32] = {
    16,  7, 20, 21, 29, 12, 28, 17,
     1, 15, 23, 26,  5, 18, 31, 10,
     2,  8, 24, 14, 32, 27,  3,  9,
    19, 13, 30,  6, 22, 11,  4, 25
}
```

Table 6.3: Key schedule permutations

```
permuted_choice1[56] = {
    57, 49, 41, 33, 25, 17,  9,
     1, 58, 50, 42, 34, 26, 18,
    10,  2, 59, 51, 43, 35, 27,
    19, 11,  3, 60, 52, 44, 36,
    63, 55, 47, 39, 31, 23, 15,
     7, 62, 54, 46, 38, 30, 22,
    14,  6, 61, 53, 45, 37, 29,
    21, 13,  5, 28, 20, 12,  4
}
```

```
permuted_choice2[48] = {
    14, 17, 11, 24,  1,  5,
     3, 28, 15,  6, 21, 10,
    23, 19, 12,  4, 26,  8,
    16,  7, 27, 20, 13,  2,
    41, 52, 31, 37, 47, 55,
    30, 40, 51, 45, 33, 48,
    44, 49, 39, 56, 34, 53,
    46, 42, 50, 36, 29, 32
}
```

6.3 Others

Specifies how many bits left to rotate the bitfields during each key schedule round.

```
rotations[16] = {
    1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1
}
```