대칭키 암호의 구현

Part 1.Ep 2: DES
YouTube: https://youtu.be/KY8vmsgT1iU
Git: https://github.com/minpie/CryptoCraftLab-minpie_public

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

발표 계획 목록

DES의 C언어 구현

계획한 향후 발표 주제 - 기존

- 1: 대칭키 암호 단일블록 암복호화의 C언어 구현 – AES, DES
- 2: OpenMPI와 AES 및 블록암호 운영모드별 병렬연산의 C언어 구현
- 3: 64비트 이상 키 길이의 공개키 암호의 C언어 구현 – RSA, Rabin, Elgamal, ECDSA

발표 계획: 24.07.19ver

- Part 1. 대칭키 암호 단일블록 C언어 구현
  - Ep1. AES
  - Ep2. DES
- Part 2. 64비트 이상 키 길이의 공개키 암호 C언어 구현
  - Ep3. GMP 라이브러리
  - Ep4. RSA 구현
  - Ep5. Rabin 구현
  - Ep6. Elgamal 구현
  - Ep7. ECDSA 구현
- Part 3. AES-운영모드 with 병렬컴퓨팅
  - Ep8. OpenMPI 라이브러리
  - Ep9. OpenMPI-AES
  - Ep10. CUDA C
  - Ep11. CUDA-AES

DES의 C언어 구현 - 개요

FIPS PUB 46-3

FEDERAL INFORMATION
PROCESSING STANDARDS PUBLICATION

Reaffirmed
1999 October 25

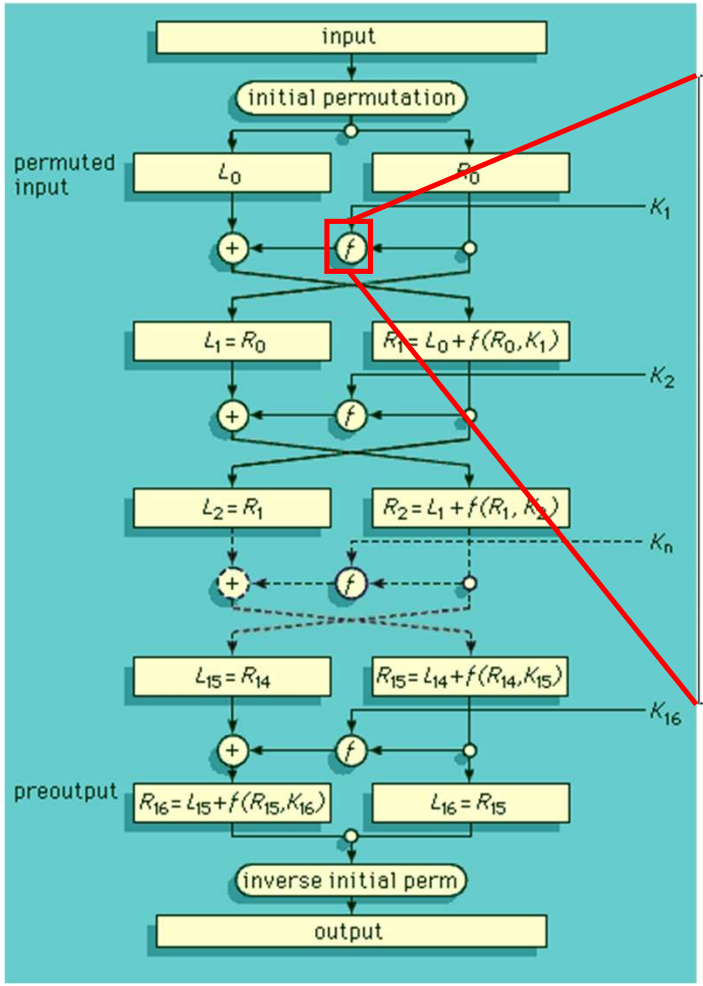
U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology

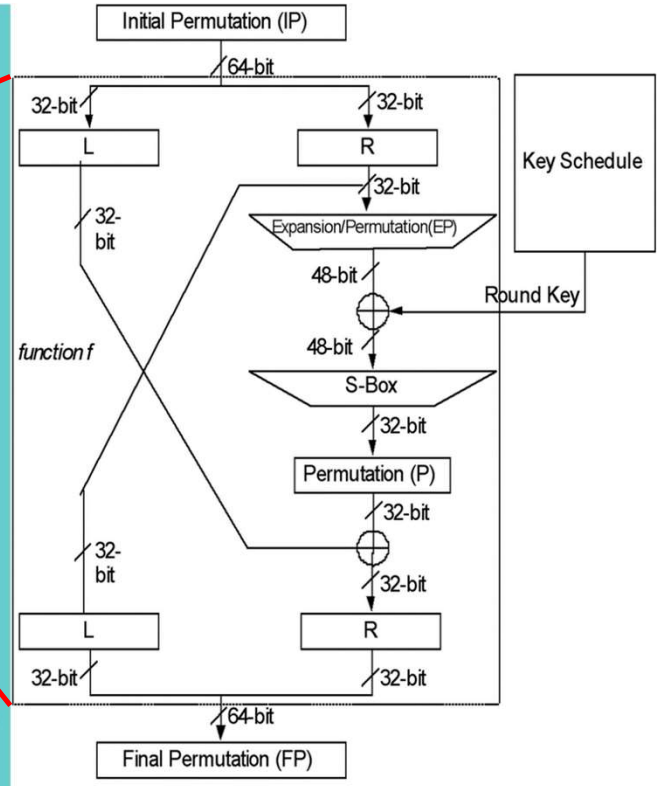
DATA ENCRYPTION STANDARD (DES)

- DES를 구현.
- 키 길이: 56비트
- 블록 길이: 64비트
- 구조: Feistel 구조
- 라운드 수: 16라운드
- NIST FIPS 46-3

https://github.com/minpie/CryptoCraftLab-minpie_public/blob/main/%EC%95%94%ED%98%B8%EA%B5%AC%ED%98%84/DataEncryptionStandard/main.c

DES의 C언어 구현 - 개요



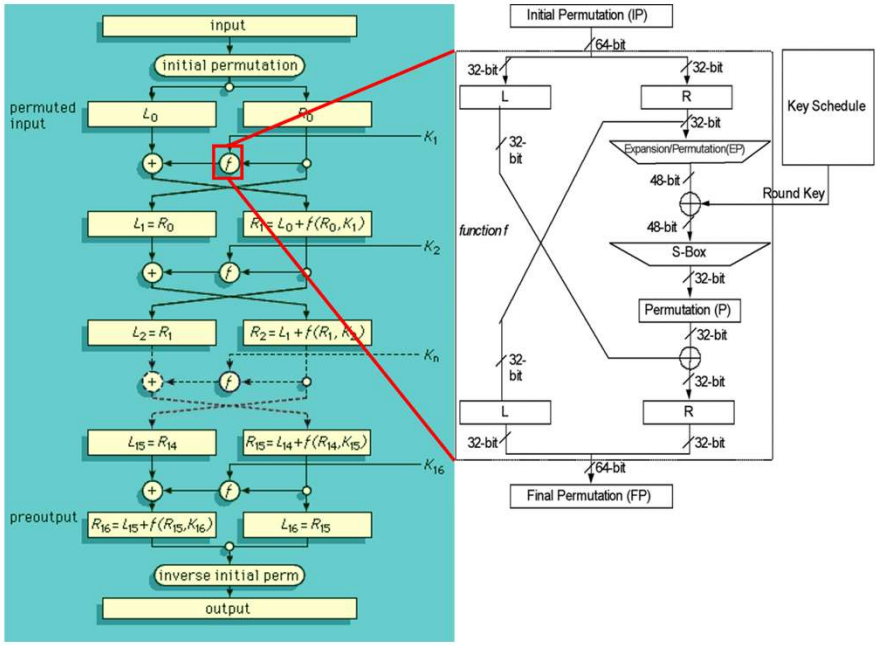


- DES를 구현.
- 키 길이: 56비트
- 블록 길이: 64비트
- 구조: Feistel 구조
- 라운드 수: 16라운드
- NIST FIPS 46-3

DES의 C언어 구현 – 단일 블록 암호화

```c
void Encryption(QWORD * Output64BitBlock, QWORD * Input64BitBlock, QWORD * RoundKeys48Bit){
    QWORD Temp64BitBlock = 0;
    // ##Calculation for Block
    InitialPermutation(&Temp64BitBlock, Input64BitBlock);
    // seperate Left/Right 32bit
    DWORD Left32Bit = 0;
    DWORD Right32Bit = 0;
    Left32Bit = (Temp64BitBlock >> 32) & 0xffffffff;
    Right32Bit = Temp64BitBlock & 0xffffffff;
    // go 16-round
    for(BYTE i=0; i<16; i++){
        QWORD Right48Bits = 0;
        DWORD TempRight32Bit = 0;
        DWORD TempRight32Bit2 = 0;
        DWORD Temp32Bit = 0;

        ExpansionPermutation(&Right48Bits, &Right32Bit); // Expansion P-Box
        Right48Bits = Right48Bits ^ RoundKeys48Bit[i]; // XOR with round key
        Substitution(&TempRight32Bit, &Right48Bits); // S-Box
        StraightPermutation(&TempRight32Bit2, &TempRight32Bit); // Straight P-Box
        Left32Bit = Left32Bit ^ TempRight32Bit2; // XOR with Left Block
        if(i != 15){
            // Swap Left-Right
            Temp32Bit = Right32Bit;
            Right32Bit = Left32Bit;
            Left32Bit = Temp32Bit;
        }else{
            Temp64BitBlock = ((QWORD)Left32Bit << 32) | Right32Bit;
        }
    }
    FinalPermutation(Output64BitBlock, &Temp64BitBlock);
}
```




```c
void Encrypt(QWORD * Output64BitBlock, QWORD * Input64BitBlock, QWORD * Initial64BitKey){
    QWORD RoundKeys48Bit[16];
    memset(&RoundKeys48Bit, 0, 128); // 128 = (64 / 8) * 16 # init to 0
    KeyExpansion(RoundKeys48Bit, Initial64BitKey);
    Encryption(Output64BitBlock, Input64BitBlock, RoundKeys48Bit);
}
typedef unsigned char BYTE;
typedef unsigned long int DWORD;
typedef unsigned long long int QWORD;
```

DES의 C언어 구현 – 단일 블록 복호화

```
272 void Decrypt(QWORD * Output64BitBlock, QWORD * Input64BitBlock, QWORD * Initial64BitKey){
273     QWORD TempRoundKeys48Bit[16];
274     QWORD RoundKeys48Bit[16];
275     memset(&TempRoundKeys48Bit, 0, 128); // 128 = (64 / 8) * 16 # init to 0
276     memset(&RoundKeys48Bit, 0, 128); // 128 = (64 / 8) * 16 # init to 0
277     KeyExpansion(TempRoundKeys48Bit, Initial64BitKey);
278
279
280     // 라운드 키 순서 뒤집기:
281     for(BYTE i=0; i<16; i++){
282         RoundKeys48Bit[i] = TempRoundKeys48Bit[15 - i];
283     }
284     Encryption(Output64BitBlock, Input64BitBlock, RoundKeys48Bit);
285 }
265 void Encrypt(QWORD * Output64BitBlock, QWORD * Input64BitBlock, QWORD * Initial64BitKey){
266     QWORD RoundKeys48Bit[16];
267     memset(&RoundKeys48Bit, 0, 128); // 128 = (64 / 8) * 16 # init to 0
268     KeyExpansion(RoundKeys48Bit, Initial64BitKey);
269     Encryption(Output64BitBlock, Input64BitBlock, RoundKeys48Bit);
270 }
```

- DES를 구현.
- 키 길이: 56비트
- 블록 길이: 64비트
- 구조: Feistel 구조
- 라운드 수: 16라운드
- NIST FIPS 46-3

DES의 C언어 구현 – KeyExpansion()

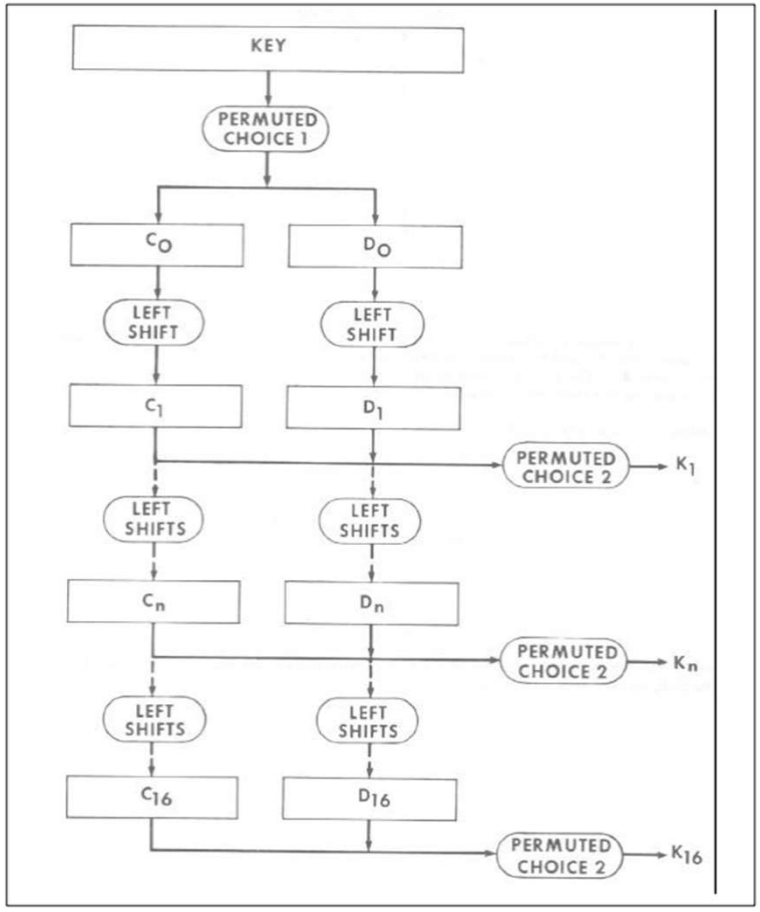


Figure 3. *Key schedule calculation*

- 총 16개의 라운드키 생성
- PC1()을 통해 64비트 초기키를 56비트 키로 변환
- PC2()를 통해 56비트 키를 48비트 라운드키로 변환

DES의 C언어 구현 – KeyExpansion()

```
149 void KeyExpansion(QWORD * Output48BitRoundKeys, QWORD * Initial64BitKey){
150     // Key Expansion
151     QWORD Key56Bit = 0;
152     DWORD Left28BitKey = 0;
153     DWORD Right28BitKey = 0;
154     BYTE ShiftAmount[16] = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};
155
156     PC1(&Key56Bit, Initial64BitKey); // PC1
157
158     Left28BitKey = Key56Bit >> 28;
159     Right28BitKey = Key56Bit & 0xfffffff;
160
161     for(BYTE i=0; i<16; i++){
162         Left28BitKey = ((Left28BitKey << ShiftAmount[i]) & 0xfffffff) | (Left28BitKey >> (28 - ShiftAmount[i])); // Circular Shift
163         Right28BitKey = ((Right28BitKey << ShiftAmount[i]) & 0xfffffff) | (Right28BitKey >> (28 - ShiftAmount[i])); // Circular Shift
164         Key56Bit = (((QWORD)Left28BitKey) << 28) | Right28BitKey; // Merge
165         PC2(&(Output48BitRoundKeys[i]), &Key56Bit); // PC2
166     }
167 }
```

Iteration Number	Number of Left Shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

- 총 16개의 라운드키 생성
- PC1()을 통해 64비트 초기키를 56비트 키로 변환
- PC2()를 통해 56비트 키를 48비트 라운드키로 변환

DES의 C언어 구현 – PC1()

```
133  void PC1(QWORD * Output56BitKey, QWORD * Input64BitKey){
134      // Permuted Choice 1
135      BYTE PC1Table[56] = {57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18, 10, 2, 59, 51, 43, 35, 27, 19,
136      for(BYTE i=0; i<56; i++){
137          *Output56BitKey = (*Output56BitKey) | ((((*Input64BitKey) >> (64 - PC1Table[i])) & 0b1) << (55 - i));
138      }
139
140  }
```

Permuted choice 1 is determined by the following table:

PC-1

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

- PC1()을 통해 64비트 초기키를 56비트 키로 변환
- 일종의 순서 바꾸기(치환)

DES의 C언어 구현 – PC2()

```
141  void PC2(QWORD * Output48BitRoundKey, QWORD * Input56BitKey){
142      // Permuted Choice 2
143      BYTE PC2Table[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,
144      for(BYTE i=0; i<48; i++){
145          *Output48BitRoundKey = (*Output48BitRoundKey) | ((((*Input56BitKey) >> (56 - PC2Table[i])) & 0b1) << (47 - i));
146      }
147  }
```

Permuted choice 2 is determined by the following table:

PC-2

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

Therefore, the first bit of K_n is the 14th bit of C_nD_n, the second bit the 17th, and so on with the 47th bit the 29th, and the 48th bit the 32nd.

- PC2()를 통해 56비트 키를 48비트 라운드키로 변환
- 일종의 순서 바꾸기(치환)

DES의 C언어 구현 – IP()

```
97   void InitialPermutation(QWORD * OutputBlock, QWORD * InputBlock){
98       BYTE IPtable[64] = {58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4, 62, 54, 46, 38
99       for(BYTE i = 0; i<64; i++){
100          *OutputBlock = (*OutputBlock) | ((((*InputBlock) >> (64 - IPtable[i])) & 0b1) << (63 - i));
101      }
102  }
```

The 64 bits of the input block to be enciphered are first subjected to the following permutation, called the initial permutation ***IP***:

IP

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

- IP()를 통해 64비트 입력을 뒤섞음.
- 일종의 순서 바꾸기(치환)

DES의 C언어 구현 – FP()

```
103  void FinalPermutation(QWORD * OutputBlock, QWORD * InputBlock){
104      BYTE FPtable[64] = {40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55, 23, 63, 31, 38, 6, 46, 14,
105      for(BYTE i = 0; i<64; i++){
106          *OutputBlock = (*OutputBlock) | ((((*InputBlock) >> (64 - FPtable[i])) & 0b1) << (63 - i));
107      }
108  }
```

IP^{-1}

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

That is, the output of the algorithm has bit 40 of the preoutput block as its first bit, bit 8 as its second bit, and so on, until bit 25 of the preoutput block is the last bit of the output.

- FP()를 통해 IP()에서 섞은 것을 원상복구
- 일종의 순서 바꾸기(치환)

DES의 C언어 구현 – ExpansionPermutation()

```
109  void ExpansionPermutation(QWORD * Output48BitBlock, DWORD * Input32BitBlock){
110      BYTE ExpansionPBox[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, 
111      for(BYTE i=0; i<48; i++){
112          *Output48BitBlock = (*Output48BitBlock) | (((QWORD)(((*Input32BitBlock) >> (32 - ExpansionPBox[i])) & 0b1)) << (47 - i)); // BIT OPERATION!!
113      }
114      // no neeeeed return
115  }
```

- 32비트 입력을 48비트로 확장

Let ***E*** denote a function which takes a block of 32 bits as input and yields a block of 48 bits as output. Let ***E*** be such that the 48 bits of its output, written as 8 blocks of 6 bits each, are obtained by selecting the bits in its inputs in order according to the following table:

E BIT-SELECTION TABLE

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

DES의 C언어 구현 – Substitution()

```
122 void Substitution(DWORD * Output32BitBlock, QWORD * Input48BitBlock){
123     // SOME OF BIT OPERATION!!!!! xD
124     *Output32BitBlock = sbox1[((*Input48BitBlock >> 46) & 0b10)| ((*Input48BitBlock >> 42) & 0b1)][(*Input48BitBlock >> 43) & 0b1111] << 4;
125     *Output32BitBlock = (*Output32BitBlock | sbox2[((*Input48BitBlock >> 40) & 0b10)| ((*Input48BitBlock >> 36) & 0b1)][(*Input48BitBlock >> 37) & 0b1111]) << 4;
126     *Output32BitBlock = (*Output32BitBlock | sbox3[((*Input48BitBlock >> 34) & 0b10)| ((*Input48BitBlock >> 30) & 0b1)][(*Input48BitBlock >> 31) & 0b1111]) << 4;
127     *Output32BitBlock = (*Output32BitBlock | sbox4[((*Input48BitBlock >> 28) & 0b10)| ((*Input48BitBlock >> 24) & 0b1)][(*Input48BitBlock >> 25) & 0b1111]) << 4;
128     *Output32BitBlock = (*Output32BitBlock | sbox5[((*Input48BitBlock >> 22) & 0b10)| ((*Input48BitBlock >> 18) & 0b1)][(*Input48BitBlock >> 19) & 0b1111]) << 4;
129     *Output32BitBlock = (*Output32BitBlock | sbox6[((*Input48BitBlock >> 16) & 0b10)| ((*Input48BitBlock >> 12) & 0b1)][(*Input48BitBlock >> 13) & 0b1111]) << 4;
130     *Output32BitBlock = (*Output32BitBlock | sbox7[((*Input48BitBlock >> 10) & 0b10)| ((*Input48BitBlock >> 6) & 0b1)][(*Input48BitBlock >> 7) & 0b1111]) << 4;
131     *Output32BitBlock = (*Output32BitBlock | sbox8[((*Input48BitBlock >> 4) & 0b10)| ((*Input48BitBlock >> 0) & 0b1)][(*Input48BitBlock >> 1) & 0b1111]);
132 }
```

Each of the unique selection functions $S_1, S_2, \ldots, S_8$, takes a 6-bit block as input and yields a 4-b block as output and is illustrated by using a table containing the recommended S_1:

S_1

Column Number

Row No.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

- 8개의 S-Box를 통해 각 6비트를 4비트로,
  즉 총 48비트를 32비트로 축소
- 전체 S-Box 표는
  Appendix 1에 존재

DES의 C언어 구현 – StraightPermutation()

```
116  void StraightPermutation(DWORD * Output32BitBlock, DWORD * Input32BitBlock){
117      BYTE StraightPBox[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};
118      for(BYTE i=0; i<32; i++){
119          *Output32BitBlock = ((*Output32BitBlock) & ~((DWORD)0b1 << (31 - i))) | (((*Input32BitBlock >> (32 - StraightPBox[i])) & 0b1) << (31 - i));
120      }
121  }
```

- 32비트 입력을 뒤섞음.

The permutation function ***P*** yields a 32-bit output from a 32-bit input by permuting the bits of the input block. Such a function is defined by the following table:

P

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

DES의 C언어 구현 – 실행 예시

```
Encryption!
Plain1          : 1122 3344 5566 7788
Plain1          : 00010001 00100010 00110011 01000100 01010101 01100110 01110111 10001000
Cipher1         : 10110101 00100001 10011110 11101000 00011010 10100111 01001001 10011101
Cipher1         : b521 9ee8 1aa7 499d
--------------------------------------------------------------------------------
Decryption!
Cipher2         : b521 9ee8 1aa7 499d
Cipher2         : 10110101 00100001 10011110 11101000 00011010 10100111 01001001 10011101
Plain2          : 00010001 00100010 00110011 01000100 01010101 01100110 01110111 10001000
Plain2          : 1122 3344 5566 7788
--------------------------------------------------------------------------------
Plain1  == Plain2  : 1
Cipher1 == Cipher2 : 1
Validation Success!
--------------------------------------------------------------------------------

############################################################
USAGE:

main.exe [input file path] [output file path] [Key] [0/1]
Key : Must 64Bit Hex!
0   : Encryption
1   : Decryption
############################################################
```

DES의 C언어 구현 – 구현 테스트 방법

[Nayuki.io: des-cipher-internals-in-excel](#)

Plaintext (input)	Key (input)	Ciphertext (output)
1122334455667788	752878397493CB70	B5219EE81AA7499D

	Message								Key schedule				
Convert to bits	0 0 0 1 0 0 0 1	0 0 1 0 0 0 1 0	0 0 1 1 0 0 1 1	0 1 0 0 0 1 0 0	0 1 0 1 0 1 0 1	0 1 1 0 0 1 1 0	0 1 1 1 0 1 1 1	1 0 0 0 1 0 0 0	0 1 1 1 0 1 0 1	0 0 1 0 1 0 0 0	0 1 1 1 1 0 0 0	0 0 1 1 1 0 0 1	0 1 1 1
Initial permutation	0 1 1 1 1 0 0 0	0 1 0 1 0 1 0 1	0 1 1 1 1 0 0 0	0 1 0 1 0 1 0 1	1 0 0 0 0 0 0 0	0 1 1 0 0 1 1 0	1 0 0 0 0 0 0 0	0 1 1 0 0 1 1 0	0 1 1 0 0 0 0 0	1 1 0 1 0 1 0 1	1 0 0 1 1 1 1 1	1 0 1 1 0 1 1 0	0 0 0 0
Round 0													
Expand right			0 1 0 0 0 0 0 0	0 0 0 0 0 0 1 1	0 0 0 0 1 1 0 1	0 1 0 0 0 0 0 0	0 0 0 0 0 0 1 1	0 0 0 0 1 1 0 1	1 1 0 0 0 0 0 1	1 0 1 0 1 0 1 1	0 0 1 1 1 1 1 1	0 1 1 0 1 1 0 0	0 0 0 0
XOR subkey			0 1 1 1 1 0 0 0	1 0 1 0 1 1 1 1	1 1 1 0 0 0 1 0	0 0 0 0 0 1 1 0	0 1 0 1 0 1 0 1	0 1 0 0 0 1 1 1	0 0 1 1 1 0 0 0	1 0 1 0 1 1 0 0	1 1 1 0 1 1 1 1	0 1 0 0 0 1 1 0	0 1 0 1
Substitute					0 1 1 1 1 0 1 1	1 1 0 0 0 1 1 0	1 1 1 0 0 0 1 0	0 1 0 1 1 0 0 0					
Permute					0 1 0 0 1 0 1 1	0 1 1 1 1 1 0 1	1 1 0 1 0 0 1 1	1 0 0 0 0 0 1 0					
XOR left	1 0 0 0 0 0 0 0	0 1 1 0 0 1 1 0	1 0 0 0 0 0 0 0	0 1 1 0 0 1 1 0	0 0 1 1 0 0 1 1	0 0 1 0 1 0 0 0	1 0 1 0 1 0 1 1	1 1 0 1 0 1 1 1					
Round 1													
Expand right			1 0 0 1 1 0 1 0	0 1 1 0 1 0 0 1	0 1 0 1 0 0 0 1	0 1 0 1 0 1 0 1	0 1 1 1 1 1 1 0	1 0 1 0 1 1 1 0	1 0 0 0 0 0 1 1	0 1 0 1 0 1 1 0	0 1 1 1 1 1 1 0	1 1 0 1 1 0 0 0	0 0 0 0
XOR subkey			0 0 0 1 0 0 1 1	1 1 0 1 0 1 1 1	1 0 0 0 0 1 0 1	0 0 0 1 1 1 0 1	1 1 1 0 0 0 1 1	1 0 1 1 1 1 0 0	1 0 0 0 1 0 0 1	1 0 1 1 1 1 1 0	1 1 0 1 0 1 0 0	0 1 0 0 1 0 0 0	1 0 0 1
Substitute					1 1 0 1 1 1 1 0	1 0 0 0 1 0 1 1	1 1 0 0 1 0 1 1	1 1 0 1 0 1 0 1					
Permute					1 1 0 1 0 0 1 1	1 1 1 1 1 1 0 0	1 0 1 0 1 0 0 1	0 1 1 1 0 0 1 1					
XOR left	0 0 1 1 0 0 1 1	0 0 1 0 1 0 0 0	1 0 1 0 1 0 1 1	1 1 0 1 0 1 1 1	0 1 0 1 0 0 1 1	1 0 0 1 1 0 1 0	0 0 1 0 1 0 0 1	0 0 0 1 0 1 0 1					

DES의 C언어 구현 – 어려웠던 부분

- Substitution() – 정확한 S-box 값
- 애매한 자료형 크기

DES의 C언어 구현 - 참고문헌

- NIST FIPS 46-3
- Nayuki.io: des-cipher-internals-in-excel

Q & A