Lab 8: DES Round Function *f* Implementation

Information Security & Cryptography(AI331)

Learning Objectives

- Understand the structure of the DES round function *f*.
- Implement *f* using the given E-expansion, S-boxes, and P-permutation.
- Test your implementation against provided known-answer test cases.

Pre-reads

Stinson, *Cryptography: Theory and Practice*, section on DES; lecture notes on the Feistel round function.

Task

In this lab, you are only required to implement the DES round function *f*. All supporting components (key schedule, E-expansion, S-boxes, and P-permutation) are provided below.

Task 1: Implement *f*(*R, K*)**:** Given a 32-bit half-block *R* and a 48-bit round subkey *K*, produce the 32-bit output using:

(a) Expansion *E*: 32 bits → 48 bits.

(b) XOR with the subkey *K*.

(c) Apply the 8 S-boxes.

(d) Apply the P-permutation.

Task 2: Test *f* on provided inputs and confirm outputs match.

1 Starter Code (C++)

You are given the key schedule, E-expansion, S-boxes, and P-permutation tables below. Your job is to complete the function body for *f*.

des.hpp

Listing 1: des.hpp

```
1 #pragma once
2 #include <cstdint>
3
4 namespace des {
5
6 // Round function f: takes 32-bit R and 48-bit subkey K (low 48 bits used) 7 uint32_t
feistel(uint32_t R, uint64_t subkey);
```



```
8
9 // Optional: expose key schedule if you want to call it from tests 10 void
```

```
key_schedule(uint64_t key64, uint64_t subkeys[16]);
} // namespace des
```

Provided tables & key schedule (authoritative)

Listing 2: des_tables.cpp (DO NOT MODIFY)

```
#include <cstdint>
#include <cstddef>

// --- Expansion E: 32 -> 48 bits ---
static const int Ebit[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
};

// --- P permutation: 32 -> 32 bits ---
static const int Pbox[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
};

// --- S-boxes: 8 boxes, each 4x16 entries ---
static const uint8_t SBOX[8][64] = {
{ // S1
14, 4,13, 1, 2,15,11, 8, 3,10, 6,12, 5, 9, 0, 7, 0,15, 7, 4,14,
2,13, 1,10, 6,12,11, 9, 5, 3, 8, 4, 1,14, 8,13, 6, 2,11,15,12, 9,
7, 3,10, 5, 0, 15,12, 8, 2, 4, 9, 1, 7, 5,11, 3,14,10, 0, 6,13
},
{ // S2
15, 1, 8,14, 6,11, 3, 4, 9, 7, 2,13,12, 0, 5,10, 3,13, 4, 7,15,
2, 8,14,12, 0, 1,10, 6, 9,11, 5, 0,14, 7,11,10, 4,13, 1, 5, 8,12,
6, 9, 3, 2,15, 13, 8,10, 1, 3,15, 4, 2,11, 6, 7,12, 0, 5,14, 9
},
{ // S3
10, 0, 9,14, 6, 3,15, 5, 1,13,12, 7,11, 4, 2, 8, 13, 7, 0, 9, 3,
4, 6,10, 2, 8, 5,14,12,11,15, 1, 13, 6, 4, 9, 8,15, 3, 0,11, 1,
2,12, 5,10,14, 7, 1,10,13, 0, 6, 9, 8, 7, 4,15,14, 3,11, 5, 2,12
},
{ // S4
7,13,14, 3, 0, 6, 9,10, 1, 2, 8, 5,11,12, 4,15, 13, 8,11, 5,
6,15, 0, 3, 4, 7, 2,12, 1,10,14, 9, 10, 6, 9, 0,12,11, 7,13,15, 1,
3,14, 5, 2, 8, 4,
```



```
3,15, 0, 6,10, 1,13, 8, 9, 4, 5,11,12, 7, 2,14
},
{ // S5
2,12, 4, 1, 7,10,11, 6, 8, 5, 3,15,13, 0,14, 9,
14,11, 2,12, 4, 7,13, 1, 5, 0,15,10, 3, 9, 8, 6,
4, 2, 1,11,10,13, 7, 8,15, 9,12, 5, 6, 3, 0,14,
```

```
11, 8,12, 7, 1,14, 2,13, 6,15, 0, 9,10, 4, 5, 3
},
{ // S6
12, 1,10,15, 9, 2, 6, 8, 0,13, 3, 4,14, 7, 5,11,
10,15, 4, 2, 7,12, 9, 5, 6, 1,13,14, 0,11, 3, 8,
9,14,15, 5, 2, 8,12, 3, 7, 0, 4,10, 1,13,11, 6,
4, 3, 2,12, 9, 5,15,10,11,14, 1, 7, 6, 0, 8,13
},
{ // S7
4,11, 2,14,15, 0, 8,13, 3,12, 9, 7, 5,10, 6, 1,
13, 0,11, 7, 4, 9, 1,10,14, 3, 5,12, 2,15, 8, 6,
1, 4,11,13,12, 3, 7,14,10,15, 6, 8, 0, 5, 9, 2,
6,11,13, 8, 1, 4,10, 7, 9, 5, 0,15,14, 2, 3,12
},
{ // S8
13, 2, 8, 4, 6,15,11, 1,10, 9, 3,14, 5, 0,12, 7,
1,15,13, 8,10, 3, 7, 4,12, 5, 6,11, 0,14, 9, 2,
7,11, 4, 1, 9,12,14, 2, 0, 6,10,13,15, 3, 5, 8,
2, 1,14, 7, 4,10, 8,13,15,12, 9, 0, 3, 5, 6,11
}
};

// --- Key schedule tables ---
// PC-1: 64 -> 56-bit key (drop parity)
static const int PC1[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
};

// PC-2: 56 -> 48-bit subkey
static const int PC2[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
};

// Left-rotation schedule for C,D (28-bit halves)
static const int ROTATIONS[16] = { 1,1,2,2,2,2,2,2, 1,2,2,2,2,2,2,1 };

// Generic permutation for a 64-bit container (MSB-first bit numbering)
static inline uint64_t permute(uint64_t in, const int* table, int n){
uint64_t out = 0;
for(int i=0;i<n;++i){
```



```
int from = table[i]-1; // 1-based -> 0-based
uint64_t bit = (in >> (64-1-from)) & 1ULL; // take bit at position 'from'
out = (out<<1) | bit;
}
return out;
}

static inline uint64_t left_rotate28(uint64_t v, int r){
v &= 0x0FFFFFFFULL; // 28 bits
```

```
115 return ((v << r) | (v >> (28 - r))) & 0x0FFFFFFFULL;
116 }
117
118 // Key schedule: input 64-bit key (with parity bits); outputs 16 subkeys (low 48 bits used)
119 void key_schedule(uint64_t key64, uint64_t subkeys[16]){
120 // PC-1: 64 -> 56 (drop parity)
121 uint64_t key56 = permute(key64, PC1, 56);
122 uint64_t C = (key56 >> 28) & 0x0FFFFFFFULL;
123 uint64_t D = key56 & 0x0FFFFFFFULL;
124 for(int r=0;r<16;++r){
125 C = left_rotate28(C, ROTATIONS[r]);
126 D = left_rotate28(D, ROTATIONS[r]);
127 uint64_t CD = (C<<28) | D; // 56 bits
128 // PC-2: 56 -> 48
129 uint64_t k48 = 0;
130 for(int i=0;i<48;++i){
131 int from = PC2[i]-1;
132 uint64_t bit = (CD >> (56-1-from)) & 1ULL;
133 k48 = (k48<<1) | bit;
134 }
135 subkeys[r] = k48; // stored in low 48 bits
136 }
137 }
138
139 // Round function prototype (implementation left to students in another file) 140 extern
uint32_t feistel(uint32_t R, uint64_t subkey);
```

des.cpp (with TODO)

Listing 3: des.cpp

```
1 #include "des.hpp"
2 #include <cstdint>
3 #include <stdexcept>
4
5 namespace des {
6
7 // E-expansion, S-boxes, P-permutation, and key schedule are provided in des_tables. cpp
8
9 uint32_t feistel(uint32_t R, uint64_t subkey) {
10 // TODO:
11 // 1. Expand 32-bit R to 48 bits using E-expansion table.
12 // 2. XOR with subkey (low 48 bits used).
13 // 3. Apply 8 S-boxes (each maps 6 bits -> 4 bits).
14 // 4. Apply P-permutation to the 32-bit result.
15 // 5. Return the 32-bit output.
```



```
16 return 0; // placeholder
17 }
18
19 } // namespace des
```

main.cpp(key scheduling output)

Listing 4: main.cpp

```cpp
#include "des.hpp"
#include <iostream>
#include <iomanip>
#include <cstdint>

int main() {
    // Example 64-bit key with parity bits (from DES standard)
    uint64_t key = 0x133457799BBCDFF1ULL;
    uint64_t subkeys[16];

    des::key_schedule(key, subkeys);

    std::cout << "Key schedule for key 0x133457799BBCDFF1\n";
    for (int i = 0; i < 16; ++i) {
        std::cout << "Round " << std::setw(2) << (i+1)
                  << ": 0x" << std::hex << std::uppercase
                  << std::setw(12) << std::setfill('0') << subkeys[i]
                  << std::dec << "\n";
    }
    return 0;
}
/*
Round 1: 0x1B02EFFC7072
Round 2: 0x79AED9DBC9E5
Round 3: 0x55FC8A06D1FA
Round 4: 0x72ADD6DB351D
Round 5: 0x7CEC07EB53A8
Round 6: 0x63A53E507B2F
Round 7: 0xEC84B7F618BC
Round 8: 0xF75E369A2D9F
Round 9: 0xDB4F1C64763C
Round 10: 0x81AC77D6ECF2
Round 11: 0x5F6A1D2F573E
Round 12: 0x0DC98C1AEDA7
Round 13: 0x33638D0CBFD0
Round 14: 0x196A0B2C95F5
Round 15: 0x42C7F0EDE5E0
Round 16: 0x7A9BE42F2009
*/
```

Correctness Tests

Verify your implementation of *f* using the following test case:

- Input *R* = 0xF0AAF0AA, Subkey *K* = 0x1B02EFFC7072, Expected output =

0x234AA9BB. 5

Submission

Submit: des.hpp, des.cpp, and a brief README.md showing test outputs. 6