Харківський університет радіоелектроніки
Факультет комп'ютерних наук
Кафедра програмної інженерії

ЗВІТ

до лабораторної роботи №1 з дисципліни
"Безпека програм та даних"
на тему "СТВОРЕННЯ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ АЛГОРИТМУ DES"

Виконав ст. гр ПЗПІ-20-2
Овчаренко Михайло Миколайович

Перевірив
асистент кафедри ПІ
Олійник О. О.

Харків 2023

МЕТА

Ознайомитись з методами і засобами симетричної криптографії, отримати навички створювання програмних засобів з використанням криптографічних інтерфейсів.

ЗАВДАННЯ

Розробити програму для шифрування та дешифрування алгоритмом DES блоків розміром 64 біт.

ХІД РОБОТИ

Реалізуємо алгоритми мовою С. Створимо функції, що відповідають за:

- генерацію ключів, - “generate_keys”;
- обчислення мережею Фейстеля, - “feistel”;
- сам алгоритм DES, що використовує вищезазначені функції та виконує шифрування та дешифрування масиву 64-бітних блоків, - “DES”.

Похідний код функцій знаходиться в додатку А коду програми, файл “main.c”.

З метою перевірки коректності роботи програми зашифруємо звичайний шматок тексту, потім дешифруємо, та перевіримо, чи збігається первинний з дешифрованим (див. рис. 1).

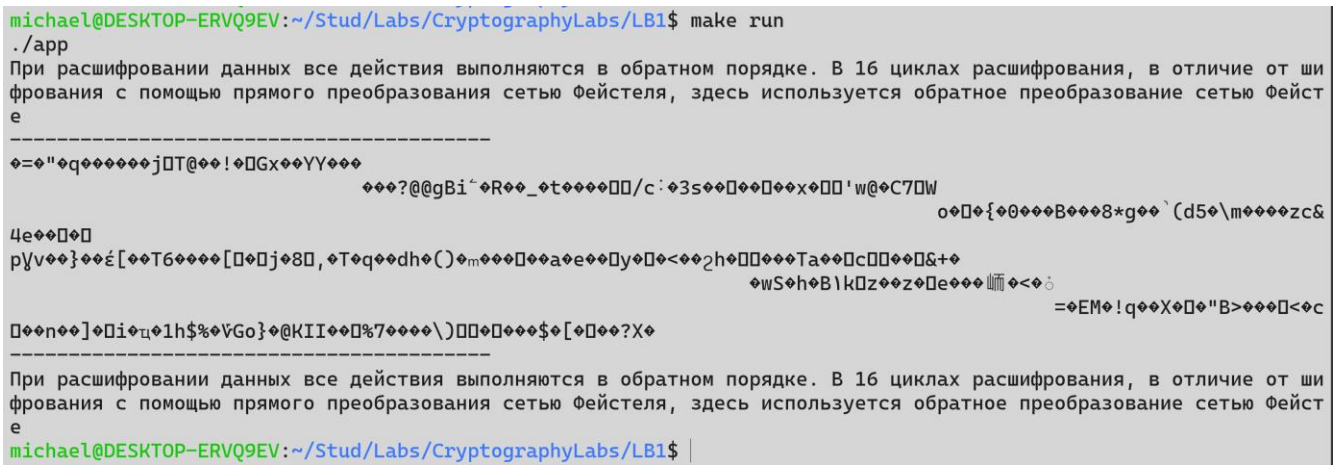

Рисунок 1 – Перевірка роботи алгоритму

Як бачимо, все працює як слід. Безглуздий набір символів посередині є зашифрованими байтами, що термінал намагається інтерпретувати як послідовність UTF-8.

ВИСНОВКИ

Розробив програму мовою C для шифрування та дешифрування даних алгоритмом DES.

ДОДАТОК А

Програмний код

Файл main.c

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  #include <stdint.h>
 4
 5  #define LB32   0x00000001
 6  #define LB64   0x0000000000000001
 7
 8  /* Inverse Initial Permutation Table */
 9  static char PI[] = {
10      40,  8, 48, 16, 56, 24, 64, 32,
11      39,  7, 47, 15, 55, 23, 63, 31,
12      38,  6, 46, 14, 54, 22, 62, 30,
13      37,  5, 45, 13, 53, 21, 61, 29,
14      36,  4, 44, 12, 52, 20, 60, 28,
15      35,  3, 43, 11, 51, 19, 59, 27,
16      34,  2, 42, 10, 50, 18, 58, 26,
17      33,  1, 41,  9, 49, 17, 57, 25
18  };
19
20   /* Post S-Box permutation */
21  static char P[] = {
22      16,  7, 20, 21,
23      29, 12, 28, 17,
24       1, 15, 23, 26,
25       5, 18, 31, 10,
26       2,  8, 24, 14,
27      32, 27,  3,  9,
28      19, 13, 30,  6,
```

```
    22, 11,  4, 25
};

static char S_[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 */
```

```
    64      12,  1, 10, 15,  9,  2,  6,  8,  0, 13,  3,  4, 14,  7,  5, 11,
    65      10, 15,  4,  2,  7, 12,  9,  5,  6,  1, 13, 14,  0, 11,  3,  8,
    66       9, 14, 15,  5,  2,  8, 12,  3,  7,  0,  4, 10,  1, 13, 11,  6,
    67       4,  3,  2, 12,  9,  5, 15, 10, 11, 14,  1,  7,  6,  0,  8, 13
    68  },{
    69      /* S7 */
    70       4, 11,  2, 14, 15,  0,  8, 13,  3, 12,  9,  7,  5, 10,  6,  1,
    71      13,  0, 11,  7,  4,  9,  1, 10, 14,  3,  5, 12,  2, 15,  8,  6,
    72       1,  4, 11, 13, 12,  3,  7, 14, 10, 15,  6,  8,  0,  5,  9,  2,
    73       6, 11, 13,  8,  1,  4, 10,  7,  9,  5,  0, 15, 14,  2,  3, 12
    74  },{
    75      /* S8 */
    76      13,  2,  8,  4,  6, 15, 11,  1, 10,  9,  3, 14,  5,  0, 12,  7,
    77       1, 15, 13,  8, 10,  3,  7,  4, 12,  5,  6, 11,  0, 14,  9,  2,
    78       7, 11,  4,  1,  9, 12, 14,  2,  0,  6, 10, 13, 15,  3,  5,  8,
    79       2,  1, 14,  7,  4, 10,  8, 13, 15, 12,  9,  0,  3,  5,  6, 11
    80  }};
    81
    82  static char feistel_expantion[] = {
    83  32, 1, 2, 3, 4, 5,
    84  4, 5, 6, 7, 8, 9,
    85  8, 9, 10, 11, 12, 13,
    86  12, 13, 14, 15, 16, 17,
    87  16, 17, 18, 19, 20, 21,
    88  20, 21, 22, 23, 24, 25,
    89  24, 25, 26, 27, 28, 29,
    90  28, 29, 30, 31, 32, 1
    91  };
    92
    93  static char key_table[] = {14,  17,     11,     24,     1,      5,      3,
28,     15,     6,     21,      10,     23,     19,     12,     4,
    94  26,     8,      16,     7,      27,     20,     13,     2,      41,     52,
31,     37,     47,    55,      30,     40,
    95  51,     45,     33,     48,     44,     49,     39,     56,     34,     53,
46,     42,     50,    36,      29,     32};
    96
```

```
 97  static char CD_shift_table[] = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2,
2, 1};
 98
 99  static char IP_table[] = {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,
100  57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3,
101  61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7};
102
103  static char C0_table[28] = {57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34,
26, 18,
104  10, 2, 59, 51, 43, 35, 27, 19, 11, 3, 60, 52, 44, 36};
105
106  static char D0_table[28] = {63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38,
30, 22,
107  14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12, 4};
108
109  int64_t initial_permutation(int64_t _64block){
110          int64_t accumulator = 0;
111          for(int i = 0; i < 64; i++){
112                  accumulator <<= 1;
113                  accumulator |= (_64block >> (64 - IP_table[i])) & LB64;
114          }
115          return accumulator;
116  }
117
118  int32_t feistel(int32_t vec, int64_t key){
119          // E(R) expantion function
120          int64_t expanded = 0; // 48 bit
121          for(int i = 0; i < 48; i++){
122                  expanded <<= 1;
123                  expanded |= (vec >> (32 - feistel_expantion[i])) & LB64;
124          }
125
126          // XOR it with key
127          int64_t B = expanded ^ key;
128
129          // S-transform
```

```
130         int32_t transformed = 0;
131         for(int i = 0; i < 8; i++){
132                 transformed <<= 4;
133                 char S = B >> ((7 - i) * 6);
134                 char a = ((S >> 4) & 2) | (S & 1); // 0..3
135                 char b = S & 0x1E; // 0..15 middle 4 bits of 6-bit S block
136                 int32_t res = S_[i][a * 16 + b]; // 4-bit value
137                 transformed |= res;
138         }
139
140         // Post S-box permutation
141         int32_t permuted = 0;
142         for(int i = 0; i < 32; i++){
143                 permuted <<= 1;
144                 permuted |= (transformed >> (32 - P[i])) & LB32;
145         }
146         return permuted;
147 }
148
149 void generate_keys(int64_t base_key, int64_t* keys){
150         // Calculate number of odd bits so that we set the lowest byte bit
151         int64_t prepared_key = 0;
152         int64_t byte_mask = 0x00000000000000FF;
153         for(int i = 0; i < 8; i++){
154                 char cur = base_key >> (i * 8);
155                 char cur2 = cur;
156                 int ones_count = 0;
157                 while (cur2){
158                         ones_count++;
159                         cur2 &= (cur2 - 1);
160                 }
161                 if (ones_count % 2 == 0){
162                         if(cur & 0x01)
163                                 cur &= 0xFE; // !0x01
164                         else
```

```
165                                     cur |= 0x01;
166                     }
167                     // set byte back
168                     prepared_key <<= 8;
169                     prepared_key |= ((int64_t) cur) & byte_mask;
170                     //base_key = !(byte_mask << (i * 8)) & base_key |
(((int64_t) cur & byte_mask) << (i * 8));
171             }
172
173             // C0 D0 Permutation
174             int32_t C0_block = 0;
175             int32_t D0_block = 0;
176             for(int i = 0; i < 28; i++){
177                     C0_block <<= 1;
178                     C0_block |= (prepared_key >> (64 - C0_table[i])) & LB32;
179
180                     D0_block <<= 1;
181                     D0_block |= (prepared_key >> (64 - D0_table[i])) & LB32;
182             }
183
184             // getting the Citer, Diter vector
185             for(int i = 0; i < 16; i++){
186                     // cyclic left shift
187                     int shift_times = CD_shift_table[i];
188                     for(int j = 0; j < shift_times; j++){
189                             C0_block = (C0_block >> 27) & LB32 | (C0_block <<
1) & 0x0fffffff;
190                             D0_block = (D0_block >> 27) & LB32 | (D0_block <<
1) & 0x0fffffff;
191                     }
192
193                     // creating common vector CiDi
194                     int64_t CiDi = ((int64_t) C0_block << 28) | D0_block;
195                     // key 48 bit
196                     int64_t key = 0;
197                     for(int j = 0; j < 48; j++){
```

```
198                         key <<= 1;
199                         key |= (CiDi >> (56 - key_table[j])) & LB64;
200                 }
201                 keys[i] = key;
202         }
203 }
204
205 void DES(int64_t* block, int64_t block_length, int64_t key, int b_enc){
206         int64_t keys[16];
207         // Generating keys
208         generate_keys(key, keys);
209
210         for(int k = 0; k < block_length; k++){
211                 // Initial permutation
212                 int64_t block_perm = initial_permutation(block[k]);
213                 int32_t L = block_perm >> 32;
214                 int32_t R = block_perm;
215                 int32_t old_R = 0, old_L = 0;
216
217                 // 16 cycles of feistel transform
218                 for(int i = 0; i < 16; i++){
219                         old_L = L;
220                         old_R = R;
221                         L = old_R;
222                         if(b_enc)
223                                 R = old_L ^ feistel(old_R, keys[i]);
224                         else
225                                 R = old_L ^ feistel(old_R, keys[15 - i]);
226                 }
227
228         //      else{ // decryption
229         //              for(int i = 15; i >= 0; i--){
230         //                      old_L = L;
231         //                      old_R = R;
232         //                      R = old_L;
```

```
233          //                      L = old_R ^ feistel(old_L, keys[i]);
234          //              }
235          //      }
236
237                  // Final reverse permutation
238                  int64_t RL_block = ((int64_t)R << 32) | ((int64_t)L &
0x00000000FFFFFFFF);
239                  int64_t accumulator = 0;
240                  for(int i = 0; i < 64; i++){
241                          accumulator <<= 1;
242                          accumulator |= (RL_block >> (64 - PI[i])) & LB64;
243                  }
244                  block[k] = accumulator;
245          }
246  }
247
248  long get_file_size(FILE* fp){
249          fseek(fp, 0, SEEK_SET);
250          fseek(fp, 0, SEEK_END);
251          long size = ftell(fp);
252          fseek(fp, 0, SEEK_SET);
253          return size;
254  }
255
256  int main(){
257
258          int64_t initial_key = 0x0123456789ABCDEF;
259
260          FILE* fp_test1 = fopen("./test/test1.txt", "rb");
261          if(fp_test1)
262          {
263                  long size = get_file_size(fp_test1);
264                  char* buf = (char*)malloc(size);
265
266                  fread(buf, size, 1, fp_test1);
```

```
267
268                 for(int i = 0; i < (size / 8) * 8; i++)
269                 {
270                         putc(buf[i], stdout);
271                 }
272                 putc('\n', stdout);
273                 puts("----------------------------------------");
274                 int64_t* start = (int64_t*) buf;
275
276                 // Cipher that text
277                 DES(start, (size / 8), initial_key, 1);
278                 for(int i = 0; i < (size / 8) * 8; i++)
279                 {
280                         putc(buf[i], stdout);
281                 }
282                 putc('\n', stdout);
283                 puts("----------------------------------------");
284
285                 // Decrypt back
286                 DES(start, (size / 8), initial_key, 0);
287                 for(int i = 0; i < (size / 8) * 8; i++)
288                 {
289                         putc(buf[i], stdout);
290                 }
291                 putc('\n', stdout);
292
293                 fclose(fp_test1);
294                 free(buf);
295         }
296
297         exit(0);
298 }
```