

AtharvK9 / Simplified-Data-Encryption-Standard-S-DES-Algorithm Public

forked from AkshayChavan7/Simplified-Data-Encryption-Standard-S-DES-Algorithm

[Code](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

Simplified-Data-Encryption-Standard-S-DES-Algorithm / SDES.cpp



AkshayChavan7 4 years ago



474 lines (385 loc) · 8.97 KB

Code

Blame



```
1  /*
2  TITLE: Program to implement S-DES Algorithm
3
4  Akshay S. Chavan          BE-B(25)
5
6  NOTE: Refer the PDF that I've kept in the same folder. It contains the information about the fu
7  */
8
9  #include<iostream>
10
11  using namespace std;
12
13
14  static int round1_key[8],round2_key[8];
15  void p10(int key[])
16  {
17      //Input:  1 2 3 4 5 6   7 8 9 10
18      //Output: 3 5 2 7 4 10  1 9 8 6
19      int out[10]={3,5,2,7,4,10,1,9,8,6};
20      int temp[10];
21
22      for(int i=0;i<10;i++)          //backup key
23          temp[i]=key[i];
24      for(int i=0;i<10;i++)
25      {
26          key[i]=temp[out[i]-1];
27          //cout<<key[i]<<"\t";
28      }
29
30  }
31
32  void p8(int key[])
33  {
34      //Input:  1 2 3 4 5 6 7   8 9 10
35      //Output: 6 3 7 4 8 5 10 9
36      int out[8]={6,3,7,4,8,5,10,9};
```

```

37     int temp[4];
38
39     for(int i=0;i<10;i++)          //backup key
40         temp[i]=key[i];
41
42     //cout<<"New key"<<endl;
43     for(int i=0;i<8;i++)
44     {
45         key[i]=temp[out[i]-1];
46         //cout<<key[i]<<"\t";
47     }
48
49 }
50
51 void p4(int s0s1[])
52 {
53     //Input:  1 2 3 4
54     //Output: 2 4 3 1
55     int out[4]={2,4,3,1};
56     int temp[4];
57
58     for(int i=0;i<4;i++)          //backup array
59         temp[i]=s0s1[i];
60
61     for(int i=0;i<4;i++)
62     {
63         s0s1[i]=temp[out[i]-1];
64     }
65
66 }
67
68 void left_shift(int left_half[], int right_half[], int shift_count)          //left shift by
69 {
70     int temp1=left_half[0];
71     int temp2=right_half[0];
72
73     for(int i=0;i<4;i++)
74     {
75         left_half[i]=left_half[i+1];
76         right_half[i]=right_half[i+1];
77     }
78     left_half[4]=temp1;
79     right_half[4]=temp2;
80
81     if(shift_count==2)
82         left_shift(left_half,right_half,1);
83
84 }
85
86 int* generate_key(int key[],int round)
87 {
88     int left_half[5],right_half[5];
89     static int key1[10],key2[8];
90     p10(key);
91     for(int i=0;i<10;i++)

```

```

91         for(int i=0; i<10; i++)
92         {
93             if(i<5)
94             {
95                 left_half[i]=key[i];
96                 //cout<<left_half[i]<<"\t";
97             }
98             else
99             {
100                 //cout<<"right"<<endl;
101                 right_half[i-5]=key[i];
102                 //cout<<right_half[i-5]<<"\t";
103             }
104         }
105
106         left_shift(left_half, right_half, 1);
107
108         for(int i=0; i<5; i++) //combine left_half and right_half to form key1
109         {
110             key1[i]=left_half[i];
111             key1[i+5]=right_half[i];
112         }
113         if(round==1)
114         {
115             p8(key1);
116             return key1;
117         }
118         else
119         {
120             left_shift(left_half, right_half, 2);
121             for(int i=0; i<5; i++) //combine left_half and right_half to form key1
122             {
123                 key2[i]=left_half[i];
124                 key2[i+5]=right_half[i];
125             }
126             p8(key2);
127             return key2;
128         }
129     }
130
131
132     void initial_permutation(int pt[])
133     {
134         //Input:  1 2 3 4 5 6 7 8
135         //Output: 2 6 3 1 4 8 5 7
136         int out[8]={2,6,3,1,4,8,5,7};
137         int temp[8];
138
139         for(int i=0; i<8; i++) //backup Plain Text Array
140             temp[i]=pt[i];
141
142         for(int i=0; i<8; i++)
143         {
144             pt[i]=temp[out[i]-1];
145         }
146     }

```

```
145         //cout<<pt[i]<<"\t";
146     }
147 }
148
149
150 void inverse_initial_permutation(int pt[])
151 {
152     //Input:  2 6 3 1 4 8 5 7
153     //Output: 1 2 3 4 5 6 7 8
154     int out[8]={2,6,3,1,4,8,5,7};
155     int temp[8];
156
157     for(int i=0;i<8;i++)          //backup Plain Text Array
158         temp[i]=pt[i];
159
160     for(int i=0;i<8;i++)
161     {
162         pt[out[i]-1]=temp[i];
163     }
164 }
165
166 int* expand_and_permute(int right_half[])
167 {
168     //Input:  1 2 3 4
169     //Output: 4 1 2 3 2 3 4 1
170     int out[8]={4,1,2,3,2,3,4,1};
171     int temp[4];
172     static int expanded_right[8];
173
174     for(int i=0;i<4;i++)          //backup Plain Text Array
175         temp[i]=right_half[i];
176
177     for(int i=0;i<8;i++)
178     {
179         expanded_right[i]=temp[out[i]-1];
180         //cout<<expanded_right[i]<<"\t";
181     }
182     return expanded_right;
183 }
184
185 int get_S0(int row,int column)
186 {
187     int s0[4][4]={
188         {01,00,11,10},
189         {11,10,01,00},
190         {00,10,01,11},
191         {11,01,11,10}
192     };
193     return s0[row][column];
194 }
195
196 int get_S1(int row,int column)
197 {
198     int s1[4][4]={
```

```

199         {00,01,10,11},
200         {10,00,01,11},
201         {11,00,01,00},
202         {10,01,00,11}
203     };
204     return s1[row][column];
205 }
206
207 int* rounds(int pt[],int key[],int round_no,int flag)
208 {
209     int left[4],right[4],*expanded_right,s0[4],s1[4],temp_key[10];
210
211     /*cout<<"\n\n Text to be decoded:\n";
212     for(int i=0;i<8;i++)
213     {
214         cout<<pt[i];
215     }*/
216
217     //cout<<"\n\nKey:";
218
219     cout<<"\nROUND-"<<round_no;
220     for(int i=0;i<10;i++)
221     {
222         //cout<<key[i]<<"\t";
223         temp_key[i]=key[i];           //backup initial key as key gets changed further
224     }
225
226     if(round_no==1)
227         initial_permutation(pt);       //step1 initial permutation of plain text
228
229
230     //cout<<"\n\nleft half:\n";
231     //divide into two halves
232     for(int i=0;i<4;i++)
233     {
234         left[i]=pt[i];
235         right[i]=pt[i+4];
236         //cout<<left[i];
237     }
238     expanded_right= expand_and_permute(right);
239
240     /*cout<<"\n\nexpanded_right:\n";
241     for(int i=0;i<8;i++)
242         cout<<expanded_right[i];
243
244     */
245     static int* key1;
246     if(flag==0)           //flag=0 is for encoding
247     {
248         key1=generate_key(key,round_no);           //key1 for round1 and key2 for
249         if(round_no==1)
250         {
251             for(int i=0;i<8;i++)
252                 round1_key[i]=key1[i];           //backup key for decoding

```

```
253         else
254         {
255             for(int i=0;i<8;i++)
256                 round2_key[i]=key1[i];
257         }
258         cout<<"\n\nEncode Key of Round "<<round_no<<endl;
259         for(int i=0;i<8;i++)
260         {
261             cout<<key1[i];
262         }
263     }
264     else //else flag=1 ie. for decoding
265     {
266         //cout<<"\n\nInside decode";
267         //for decoding we use the keys in reverse order
268
269         if(round_no==1) //if round1 use key2
270         {
271             //cout<<"\n Inside round1";
272             for(int i=0;i<8;i++)
273             {
274                 key1[i]=round2_key[i];
275                 //cout<<round2_key[i];
276                 //cout<<"test";
277             }
278         }
279         else //if round2 use key1
280         {
281             //cout<<"\n Inside round2";
282             for(int i=0;i<8;i++)
283             {
284                 //cout<<round1_key[i];
285                 key1[i]=round1_key[i];
286             }
287         }
288
289         cout<<"\n\nDecode Key of Round "<<round_no<<endl;
290         for(int i=0;i<8;i++)
291         {
292             cout<<key1[i];
293         }
294     }
295     /*cout<<"\n\nExpanded right\n";
296     for(int i=0;i<8;i++)
297     {
298         cout<<expanded_right[i]<<"\t";
299     }
300     cout<<"\n\n";*/
301
302     for(int i=0;i<8;i++)
303     {
304         expanded_right[i]=expanded_right[i] ^ key1[i];
305         if(i<4)
306             s0[i]=expanded_right[i];
```

```

307         else
308             s1[i-4]=expanded_right[i];
309     }
310
311     int row=s0[3]+(s0[0]*2);           //step 4
312     int column=s0[2]+(s0[1]*2);
313     static int s0s1[4];
314     int ss0=get_S0(row,column);
315     //cout<<"\nRow: "<<row<<"Column: "<<column;
316     row=s1[3]+(s1[0]*2);
317     column=s1[2]+(s1[1]*2);
318     //cout<<"\nRow: "<<row<<"Column: "<<column;
319     int ss1=get_S1(row,column);
320
321     s0s1[1]=ss0%10;
322     s0s1[0]=ss0/10;
323     s0s1[3]=ss1%10;
324     s0s1[2]=ss1/10;
325
326     /*cout<<"\n\nBefore P4:\n";
327     for(int i=0;i<4;i++)
328         cout<<s0s1[i];
329     */
330     p4(s0s1);
331
332     static int new_plain_text[8];
333     //s0s1 EXOR Left_Half from step 1
334     for(int i=0;i<4;i++)
335     {
336         s0s1[i]=s0s1[i] ^ left[i];
337         //swap the s0s1 and right half from step 1 to generate plain text for next round
338         if(round_no!=2)           //if round is not 2nd one and it's not for decoding
339         {
340             new_plain_text[i]=right[i];
341             new_plain_text[i+4]=s0s1[i];
342         }
343         else                       //else don't swap
344         {
345             new_plain_text[i+4]=right[i];
346             new_plain_text[i]=s0s1[i];
347         }
348     }
349
350     /*cout<<"\n\ns0s1:\n";
351     for(int i=0;i<4;i++)
352         cout<<s0s1[i];
353     */
354     cout<<"\n\nRound "<<round_no<<" Output:\n";
355     for(int i=0;i<8;i++)
356         cout<<new_plain_text[i]<<"\t";
357     cout<<endl;
358
359
360

```

```

361         if(round_no==1)
362         {
363             //cout<<"\n\ngoing for next round\n";
364             if(flag==0) //if encoding
365                 rounds(new_plain_text,temp_key,2,0);
366             else //else decoding
367                 rounds(new_plain_text,temp_key,2,1);
368         }
369         else
370         {
371             return new_plain_text;
372         }
373     }
374 }
375
376 int* encode(int pt[],int* round_text,int key[])
377 {
378     round_text=rounds(pt,key,1,0);
379     inverse_initial_permutation(round_text);
380
381     cout<<"\n\n-----FINAL CIPHER TEXT-----\n";
382     for(int i=0;i<8;i++)
383         cout<<round_text[i];
384
385     return round_text;
386 }
387
388 void decode(int pt[], int* cipher_text,int key[])
389 {
390     int *new_ct=rounds(cipher_text,key,1,1); //flag=1 for decoding
391     inverse_initial_permutation(new_ct);
392
393     cout<<"\n\n-----DECODED TEXT-----\n";
394     for(int i=0;i<8;i++)
395         cout<<new_ct[i];
396 }
397 int main()
398 {
399     int *round_text, *cipher_text, pt[8],key[10];
400     cout<<"\nEnter the plain text (8-bits) :";
401     for(int i=0;i<8;i++)
402         cin>>pt[i];
403     cout<<"\nEnter the key (10-bits) :";
404     for(int i=0;i<10;i++)
405         cin>>key[i];
406
407     //int pt[8]={0,1,1,1,0,0,1,0};
408     //int key[10]={1,0,1,0,0,0,0,0,1,0};
409
410     cout<<"\n-----ENCRYPTION-----\n";
411     cipher_text=encode(pt,round_text,key); //Encryption
412     cout<<"\n\n\n-----DECRYPTION-----\n";
413     decode(pt,cipher_text,key); //Decryption
414     return 0;

```



```
415     }
416
417     /*
418     OUTPUT:
419
420     C:\Users\Akshay Chavan\Desktop>g++ SDES.cpp
421
422     C:\Users\Akshay Chavan\Desktop>a
423
424     Enter the plain text (8-bits) :0 1 1 1  0 0 1 0
425
426     Enter the key (10-bits) :1 0 1 0 0 0 0 1 0
427
428     -----ENCRYPTION-----
429
430     ROUND-1
431
432     Encode Key of Round 1
433     10100100
434
435     Round 1 Output:
436     1      0      0      1      1      1      0      1
437
438     ROUND-2
439
440     Encode Key of Round 2
441     01000011
442
443     Round 2 Output:
444     1      1      1      0      1      1      0      1
445
446
447     -----FINAL CIPHER TEXT-----
448     01110111
449
450
451     -----DECRYPTION-----
452
453     ROUND-1
454
455     Decode Key of Round 1
456     01000011
457
458     Round 1 Output:
459     1      1      0      1      1      0      0      1
460
461     ROUND-2
462
463     Decode Key of Round 2
464     10100100
465
466     Round 2 Output:
467     1      0      1      0      1      0      0      1
468
```

```
469
470  -----DECODED TEXT-----
471  01110010
472  C:\Users\Akshay Chavan\Desktop>
473
474  */
```