

**Ecole supérieure en science et technologie de l'informatique  
et du numérique ESTIN**

---

## **Compte rendu TP 01**

**Mini DES**

---

**Réalisé par :**

**Benmedakhene Souha (G2) .**

**Encadré par :**

**Dr . Bouchoucha Lydia**

**Année universitaire : 2023/2024**

# Introduction :

La sécurité des données est un enjeu majeur dans le monde numérique actuel. Le chiffrement est une technique essentielle pour protéger la confidentialité des informations sensibles lors de leur transmission ou stockage. Dans ce contexte, le TP visait à explorer et à implémenter l'algorithme de chiffrement Mini DES, une version simplifiée du célèbre standard de chiffrement DES (Data Encryption Standard).

Le Mini DES utilise une clef de **10 bits** pour chiffrer des blocs de texte de **8 bits**.

## Objectifs :

1. Comprendre les principes fondamentaux du chiffrement :
  - Appréhender les concepts de base du chiffrement, y compris la substitution, la permutation et les opérations logiques.
  - Comprendre l'importance du chiffrement dans la protection des données sensibles contre les accès non autorisés.
2. Acquérir une expérience pratique dans l'implémentation d'un algorithme de chiffrement :
  - Se familiariser avec le processus de traduction des principes théoriques du chiffrement en code informatique.
  - Renforcer ses compétences en programmation en utilisant un langage de programmation tel que **Python** pour mettre en œuvre un algorithme de chiffrement.
3. Explorer les différentes étapes du processus de chiffrement Mini DES :
  - Analyser les étapes de **génération de clés**, de permutation initiale, de substitution, etc.
  - Comprendre comment ces étapes contribuent à la sécurité et à la robustesse de l'algorithme de chiffrement.
4. Tester le chiffrement en utilisant des exemples concrets :
  - Utiliser des exemples de textes en clair et de clés pour tester l'implémentation de l'algorithme de chiffrement.

# Implémentation :

## Etape 1: Génération des clés

L'étape de génération des clés dans l'algorithme Mini DES est cruciale car elle permet de créer les clés nécessaires pour chiffrer et déchiffrer les données. Cette étape se divise en plusieurs sous-étapes :

### 1. Génération de la clé initiale (K0) :

- La clé initiale K0 est fournie en entrée ou générée aléatoirement. Dans Mini DES, la clé est représentée par une séquence de 10 bits.
- Cette clé initiale est ensuite soumise à une permutation spécifique appelée permutation initiale (P10). Cette permutation réarrange les bits de la clé initiale selon une table de permutation prédéfinie. Le résultat est une clé intermédiaire de 10 bits .

### 2. Dérivation des clés K1 et K2 :

- Une fois que la clé initiale est obtenue, elle est divisée en deux parties égales, généralement désignées comme étant L0 et R0. Chaque partie contient 5 bits.
- Ensuite, chaque partie subit un décalage circulaire vers la gauche. Pour K1, L0 et R0 sont décalés d'un bit vers la gauche, tandis que pour K2, ils sont décalés de deux bits vers la gauche.
- Après les décalages, les deux parties sont réassemblées et soumises à une autre permutation spécifique appelée permutation de choix (P8). Cette permutation réduit la taille de la clé de 10 bits à 8 bits en réorganisant les bits selon une table de permutation prédéfinie.
- Le résultat final est la clé de chiffrement K1 pour le premier round de chiffrement, et la clé K2 pour le second round.

**Et voici le code en Python :**

```

mini_DES > generate_keys
1 # Permutation box P10
2 P10 = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]
3
4 # Permutation box P8
5 P8 = [6, 3, 7, 4, 8, 5, 10, 9]
6
7 def permute(key, permutation_box):
8     permuted_key = [key[i - 1] for i in permutation_box]
9     return ''.join(permuted_key)
10
11 def left_circular_shift(bits):
12     return bits[1:] + bits[0]
13
14 def generate_keys(K0):
15     # Apply P10 permutation
16     K0_permuted = permute(K0, P10)
17
18     # Split K0 into two parts
19     L0 = K0_permuted[:5]
20     R0 = K0_permuted[5:]
21
22     # Left circular shift on L0 and R0 for K1
23     L1 = left_circular_shift(L0)
24     R1 = left_circular_shift(R0)
25
26     # Concatenate L1 and R1 for K1
27     result1 = L1 + R1
28
29     # Apply P8 permutation to get K1
30     K1 = permute(result1, P8)
31
32     # Left circular shift on L1 and R1 by 2 bits for K2
33     L2 = left_circular_shift(left_circular_shift(L1))
34     R2 = left_circular_shift(left_circular_shift(R1))
35
36     # Concatenate L2 and R2 for K2
37     result2 = L2 + R2
38
39     # Apply P8 permutation to get K2
40     K2 = permute(result2, P8)
41
42     return K1, K2
43
44 # Take input for K0
45 K0 = input("Enter the 10-bit key K0: ")
46
47 # Generate K1 and K2
48 K1, K2 = generate_keys(K0)
49
50 print("K1:", K1)
51 print("K2:", K2)
52

```

## Etape 2: Chiffrement

L'étape de chiffrement dans l'algorithme Mini DES est le cœur du processus, où les données en clair sont transformées en données chiffrées à l'aide des clés de chiffrement générées précédemment (K1 et K2). Cette étape est composée de plusieurs sous-étapes, chacune contribuant à la sécurité et à la confidentialité des données chiffrées. Voici une explication détaillée de chaque aspect de l'étape de chiffrement :

### 1. Permutation initiale (PI) :

- Au début de l'étape de chiffrement, les données en clair sont soumises à une permutation initiale (PI). Cette permutation réorganise les bits du texte en clair selon une table de permutation spécifique appelée permutation initiale (PI). L'objectif de cette permutation est de mélanger les bits du texte en clair de manière à améliorer la diffusion et la confusion, deux propriétés essentielles pour renforcer la sécurité du chiffrement.

## 2. **Rounds de chiffrement** (en utilisant la fonction $F(K_i)$ ) :

- Le chiffrement dans Mini DES se déroule en deux rounds, chacun utilisant une clé de chiffrement différente ( $K_1$  pour le premier round et  $K_2$  pour le deuxième round).
- Dans chaque round, les blocs L et R sont soumis à une série d'opérations, y compris l'expansion, la substitution, la permutation et le XOR avec la clé de chiffrement correspondante.
- L'expansion consiste à étendre le bloc R0 en utilisant une table de permutation spécifique (EP), qui ajoute des bits supplémentaires au bloc pour augmenter la complexité du chiffrement.
- Ensuite, le bloc étendu est combiné avec la clé de chiffrement ( $K_1$  ou  $K_2$ ) en utilisant l'opération de XOR (OU exclusif).
- Après le XOR, le résultat est divisé en deux parties, qui sont ensuite soumises à des S-boxes (boîtes de substitution) . Les S-boxes remplacent chaque groupe de bits par un autre groupe de bits en utilisant des tables de substitution spécifiques ( $S_0$  pour la première partie et  $S_1$  pour la deuxième partie).
- Les résultats des S-boxes sont permutés à l'aide d'une autre table de permutation spécifique ( $P_4$ ), ce qui réorganise les bits pour obtenir le texte chiffré.
- Après on va faire un XOR entre le résultat et le " L " en entrée .
- Enfin , à la fin de  $F(K_1)$  on va faire un SW ( les blocs de sortie sont combinés dans un ordre spécifique) et la même chose pour  $F(K_2)$  .

### 3. Permutation finale (PI -1) :

- Une fois les rounds de chiffrement terminés, les blocs de sortie sont soumis à une permutation finale (PI -1) ( c'est la matrice inverse de PI). Cette permutation inverse la permutation initiale et restaure l'ordre original des bits, produisant ainsi le texte chiffré final.

### Et voici le code en Python :

```
48
49 PI = [2, 6, 3, 1, 4, 8, 5, 7]
50
51 def initial_permutation(text):
52     return permute(text, PI)
53
54
55 EP = [4, 1, 2, 3]
56
57 S0 = [
58     ['01', '00', '11', '10'],
59     ['11', '10', '01', '00'],
60     ['00', '10', '01', '11'],
61     ['11', '01', '00', '10']
62 ]
63
64 S1 = [
65     ['00', '10', '10', '11'],
66     ['10', '00', '01', '11'],
67     ['11', '00', '01', '00'],
68     ['10', '01', '00', '11']
69 ]
70
71 P4 = [2, 4, 3, 1]
72
73 def xor(a, b):
74     # Perform bitwise XOR between two binary strings of equal length
75     return ''.join(str(int(x) ^ int(y)) for x, y in zip(a, b))
76
77
78 def s_box(bits, sbox):
79     row = int(bits[0] + bits[3], 2)
80     col = int(bits[1] + bits[2], 2)
81     return sbox[row][col]
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```

120
121
122 def encrypt(plaintext, K1, K2):
123     # Initial permutation
124     permuted_text = initial_permutation(plaintext)
125     # Split into L0 and R0
126     L0 = permuted_text[:4]
127     R0 = permuted_text[4:]
128     # Round 1
129     R1 = f(R0, K1)
130     # Swap and permute
131     swapped_text = swap_and_permute(R0, R1)
132     # Round 2
133     R2 = f_with_k2(swapped_text[:4], K2)
134     # Swap and permute final
135     final_text = swap_and_permute_final(R2, swapped_text[4:])
136     # Final permutation
137     encrypted_text = final_permutation(final_text)
138     return encrypted_text
139
140
141 plaintext = input("Enter the plaintext (8 bits): ")
142
143
144 # Encrypt and print the result
145 encrypted_text = encrypt(plaintext, K1, K2)
146 print("Encrypted text:", encrypted_text)

```

## Etape 3: Test et Validation

La validation est une étape critique dans le processus de développement d'un algorithme de chiffrement, car elle permet de s'assurer que le chiffrement fonctionne comme prévu et offre un niveau de sécurité adéquat .

Pour valider le chiffrement, différents exemples de texte en clair et de clés sont utilisés pour tester l'implémentation.

Text en claire : 011110010 et la clé K0 : 1010000010 .

```

Encrypted text: 011110010
PS C:\Users\HP\Desktop\new> python -u "c:\Users\HP\Desktop\new\mini_DES"
Enter the 10-bit key K0: 

```

```

PS C:\Users\HP\Desktop\new> python -u "c:\Users\HP\Desktop\new\mini_DES"
Enter the 10-bit key K0: 1010000010
Enter the plaintext (8 bits): 

```

```
PS C:\Users\HP\Desktop\new> python -u "c:\Users\HP\Desktop\new\mini_DES"
Enter the 10-bit key K0: 1010000010
Enter the plaintext (8 bits): 01110010
K1: 10100100
K2: 01000011
Encrypted text: 01110111
PS C:\Users\HP\Desktop\new> █
```

## Conclusion :

Dans le cadre de ce TP, nous avons eu l'opportunité d'approfondir notre compréhension des concepts fondamentaux du chiffrement, ainsi que de nous familiariser avec l'algorithme Mini DES. En mettant en œuvre cet algorithme de manière pratique, nous avons acquis des connaissances approfondies sur son fonctionnement interne, notamment sur les différentes étapes telles que la génération des clés .. Cette expérience pratique nous a permis de mieux appréhender les défis techniques associés à l'implémentation d'un algorithme de chiffrement, notamment en ce qui concerne la manipulation des données binaires et la gestion des opérations de permutation et de substitution.