Technische Universität München
Fakultät für Informatik
Prof. Dr. A. Knoll

SS 2006
Übungsblatt 4
16. Juni 2006

Übungen zu Einführung in die Informatik II

Aufgabe 12 DES Verfahren

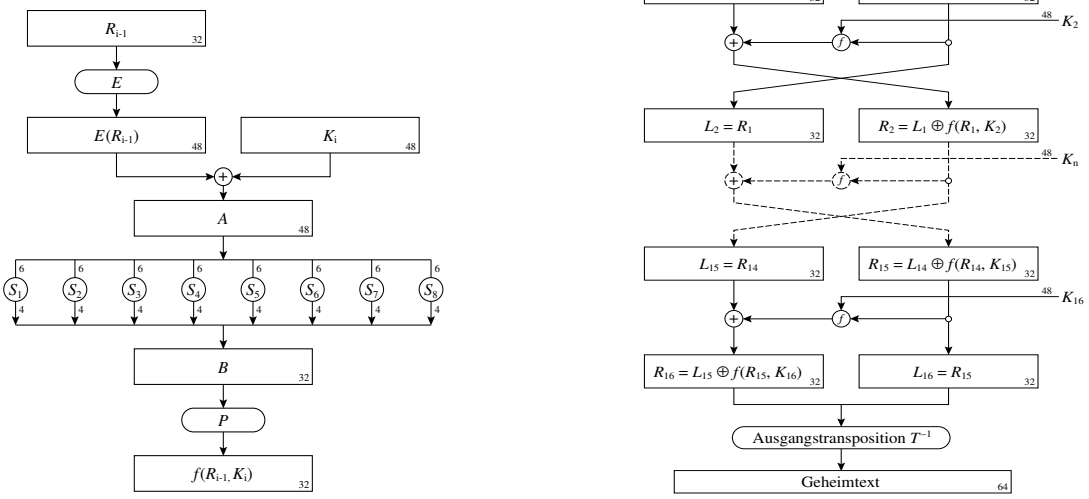
Beim DES-Verfahren wird ein Nachrichtenblock von 64 Bit Länge mit Hilfe eines Schlüssels von 56 Bit Länge chiffriert. Nach einer Eingangstransposition T wird der Klartext aufgeteilt in zwei Hälften L_0 und R_0 von je 32 Bit Länge. Darauf folgen 16 'Runden' mit

$$L_i = R_{i-1} \quad \text{und} \quad R_i = L_{i-1} \oplus f(R_{i-1}, K_i) \qquad i = 1, \ldots, 16$$

wobei die Schlüssel K_i Teilmengen der Bits des ursprünglichen Schlüssels sind.

a) Zeigen Sie, dass die Dechiffrierung $DES_K^{-1}(M)$ einer Chiffrierung $DES_K(M)$ mit den Schlüsseln K_i in umgekehrter Reihenfolge entspricht.

b) Zeigen Sie, dass gilt: $DES_{\overline{K}}(\overline{M}) = \overline{DES_K(M)}$
wobei $\overline{x}$ das bitweise Komplement
von x bezeichnet.



c) Führen Sie nun die erste Runde der DES-Verschlüsselung unter Verwendung der nachfolgenden Tabellen per Hand aus (mit selbst gewähltem Schlüssel und Klartext).

d) Implementieren Sie den DES-Algorithmus mit Hilfe der Tabellen in Java (Sie können zur Vereinfachung auch selbst gewählte Zufallstabellen verwenden. Sender und Empfänger der Nachricht müssen aber natürlich die selben Tabellen verwenden. Diese sind daher in der Spezifikation von DES eindeutig festgelegt. Die unten aufgeführten Tabellen sind dagegen zufällig erzeugt worden und enthalten damit evtl. Schwachstellen).

DES-Tabellen (die jeweilige Bitnummer der Ergebniszeichenfolge ist *kursiv* gesetzt. Die darunter stehende Zahl gibt an welches Bit der Ausgangszeichenfolge an diese Stelle kopiert wird):

Eingangstransposition T ($\Rightarrow$ Ausgangstransposition T^{-1})

63	*62*	*61*	*60*	*59*	*58*	*57*	*56*	*55*	*54*	*53*	*52*	*51*	*50*	*49*	*48*	*47*	*46*	*45*	*44*	*43*	*42*	*41*	*40*	*39*	*38*	*37*	*36*	*35*	*34*	*33*	*32*
22	57	29	44	20	37	62	8	23	56	7	39	5	21	61	14	28	60	11	27	43	58	10	41	9	25	40	55	6	38	52	4
31	*30*	*29*	*28*	*27*	*26*	*25*	*24*	*23*	*22*	*21*	*20*	*19*	*18*	*17*	*16*	*15*	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
35	51	3	19	34	50	2	17	33	49	0	16	32	47	63	15	31	46	45	12	26	24	53	18	1	30	13	42	36	48	59	54

Schlüsseltransposition (damit wird aus dem ursprünglichen 64bit-Schlüssel ein 56bit-Schlüssel erzeugt; die Paritätsbits 0, 8, 16 ... 56 dienen dabei als Checksumme modulo 2 der jeweils nachfolgenden Bits und werden danach nicht mehr verwendet):

55	*54*	*53*	*52*	*51*	*50*	*49*	*48*	*47*	*46*	*45*	*44*	*43*	*42*	*41*	*40*	*39*	*38*	*37*	*36*	*35*	*34*	*33*	*32*	*31*	*30*	*29*	*28*
25	47	41	57	35	63	13	29	44	59	11	30	42	58	10	9	23	39	54	6	22	38	53	5	21	46	52	61
27	*26*	*25*	*24*	*23*	*22*	*21*	*20*	*19*	*18*	*17*	*16*	*15*	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
4	19	51	3	18	34	50	1	17	33	31	15	62	45	60	12	28	27	7	55	37	36	2	49	14	43	26	20

danach wird der jeweilige Rundenschlüssel K_i gewonnen, indem der 56bit Schlüssel in zwei 28bit Teile zerlegt wird. Die beiden Teile werden dann zyklisch verschoben. In geraden Runden (2, 4 ... 16) um jeweils ein bit, in den restlichen Runden um zwei bits.

Die oben definierte Funktion f verknüpft einen 48bit-Wert, der aus der rechten Seite R_{i-1} gewonnen wird (siehe unten) mit dem Rundenschlüssel K_i. Dazu muss dieser auf 48bit komprimiert werden, was mit folgender Transposition geschieht:

47	*46*	*45*	*44*	*43*	*42*	*41*	*40*	*39*	*38*	*37*	*36*	*35*	*34*	*33*	*32*	*31*	*30*	*29*	*28*	*27*	*26*	*25*	*24*
31	2	14	30	8	41	49	16	32	48	15	47	45	13	43	10	26	42	9	24	39	55	0	23
23	*22*	*21*	*20*	*19*	*18*	*17*	*16*	*15*	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
7	22	38	6	21	28	11	25	54	5	37	53	4	20	34	50	18	33	1	40	52	35	17	46

Um diesen Schlüssel nun mit R_{i-1} zu verknüpfen, muss R_{i-1} noch auf 48bit expandiert werden. Dies geschieht mit folgender Expansionstransposition (wie man sieht wird R_{i-1} in Viererblocks zerlegt, die jeweils durch die bits des benachbarten Blocks zu Sechserblocks aufgefüllt werden):

47	*46*	*45*	*44*	*43*	*42*	*41*	*40*	*39*	*38*	*37*	*36*	*35*	*34*	*33*	*32*	*31*	*30*	*29*	*28*	*27*	*26*	*25*	*24*
0	31	30	29	28	**27**	**28**	27	26	25	24	**23**	**24**	23	22	21	20	**19**	**20**	19	18	17	16	**15**
23	*22*	*21*	*20*	*19*	*18*	*17*	*16*	*15*	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
16	15	14	13	12	**11**	**12**	11	10	9	8	**7**	**8**	7	6	5	4	**3**	**4**	3	2	1	0	**31**

Das durch die XOR-Verknüpfung von Rundenschlüssel K_i und expandiertem R_{i-1} erhaltene Ergebnis A (ebenfalls 48bit) wird nun in acht Sechserblocks zu jeweils sechs bit zerlegt. Das erste und letzte bit dieses Blocks wird zu einem 2bit Wert x_i zusammengezogen, der restliche 4bit-Block codiert den Wert y_i (i ist die Blocknummer). Für jeden der acht Sechserblocks von A existiert nun eine eigene Tabelle (so genannte S_i-Box). In dieser wird mit Hilfe von x_i (Zeile) und y_i (Spalte) eine 4bit Zahl (n_i) selektiert. Die acht Werte n_0 bis n_7 werden anschließend zu einer 32bit Zahl (B) konkateniert. Die S-Boxen sehen dabei wie folgt aus:

S_0

9	11	8	7	1	12	10	6	5	4	3	2	0	15	14	13
6	2	1	9	14	13	12	11	10	8	7	5	4	3	0	15
9	6	5	10	12	2	1	0	15	14	13	11	8	7	4	3
2	0	13	6	4	1	15	14	12	11	10	9	8	7	5	3

S_1

12	2	15	14	7	8	4	3	1	6	5	0	13	11	10	9
15	13	10	9	3	14	12	11	8	7	6	5	4	2	1	0
15	14	9	2	5	6	4	3	1	0	13	12	11	10	8	7
6	2	14	7	10	9	12	8	5	4	3	1	0	15	13	11

S_2

0	11	3	1	12	10	15	14	13	9	8	7	6	5	4	2
12	1	6	13	2	0	15	14	11	10	9	8	7	5	4	3
3	5	2	12	14	7	6	4	1	0	15	13	11	10	9	8
13	15	11	5	0	14	12	10	9	8	7	6	4	3	2	1

S_3

7	9	4	3	13	5	8	6	2	1	0	15	14	12	11	10
10	2	14	8	7	3	1	0	15	13	12	11	9	6	5	4
5	7	3	2	14	15	13	1	12	11	10	9	8	6	4	0
7	12	8	4	13	11	10	9	6	5	3	2	1	0	15	14

S_4

1	4	0	11	2	7	6	5	3	13	15	14	12	10	9	8
15	8	5	4	0	11	10	9	7	6	3	2	1	14	13	12
14	4	5	15	10	9	11	1	0	3	13	12	8	7	6	2
10	12	7	6	14	8	11	9	5	4	3	2	1	0	15	13

S_5

9	8	4	12	15	11	10	7	6	5	3	2	1	0	14	13
13	6	3	2	12	9	8	7	5	4	1	0	15	14	11	10
15	10	5	3	12	2	9	8	7	11	6	4	1	0	14	13
2	5	1	12	10	14	13	11	9	8	7	6	4	3	0	15

S_6

6	15	11	10	3	5	8	7	4	2	1	0	14	13	12	9
14	8	4	3	12	15	13	0	1	11	10	9	7	6	5	2
14	10	4	12	0	15	13	11	9	8	7	6	5	3	2	1
2	6	9	13	10	12	7	5	4	11	3	8	1	0	15	14

S_7

10	3	0	6	8	13	5	4	2	1	15	14	9	12	11	7
8	6	2	1	12	4	15	14	13	11	10	9	7	5	3	0
9	13	5	4	14	10	8	7	12	6	3	2	1	0	11	15
10	5	2	14	12	4	3	13	15	11	8	9	7	6	1	0

Zum Abschluss der Runde wird das so gewonnen Ergebnis noch einmal durch folgende Transposition (P) permutiert und danach mit der linken Seite L_{i-1} verknüpft (XOR):

31	*30*	*29*	*28*	*27*	*26*	*25*	*24*	*23*	*22*	*21*	*20*	*19*	*18*	*17*	*16*	*15*	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
11	19	30	14	24	0	26	10	25	29	9	8	7	23	22	6	5	21	4	20	3	2	18	1	17	16	31	15	13	28	12	27

Aufgabe 13 Threads in Java

Nebenläufige Vorgänge in Java können mit Hilfe der abstrakten Klasse `Thread` entworfen werden. Hierzu muss von dieser Klasse abgeleitet werden und die Methode `run` implementiert werden. Der Thread kann anschließend über die Methode `start` gestartet werden.

```
import java.lang.Thread;

public class MyThread extends Thread {
   ...
   public void run() {
      ...
   }
   ...
}
```

a) Implementieren Sie einen Thread, dessen Funktionalität darin besteht einen übergebenen Text auf der Konsole in einer Endlosschleife auszugeben. Erzeugen Sie anschließend drei Instanzen dieses Threads mit den Texten *tick*, *tack* und *tock* und lassen Sie diese parallel laufen.

b) Informieren Sie sich über die Funktionalität der Methode `yield`. Erweitern Sie Ihren ursprünglichen Ausgabe-Thread um den Aufruf dieser Methode und vergleichen Sie die Ausgaben miteinander.

c) Informieren Sie sich über die Funktionalität der Methode `sleep`. Erweitern Sie Ihren ursprünglichen Ausgabe-Thread um den Aufruf dieser Methode und vergleichen Sie die Ausgaben miteinander.

Aufgabe 14 Matrix-Multiplikation mit Threads in Java

Verwenden Sie für die nachfolgenden Aufgaben Java-Arrays zur Darstellung von Matrizen (z.B. `float[][] a`).

a) Wiederholen Sie die Multiplikation von Matrizen anhand von folgendem Beispiel:

$$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{pmatrix}$$

b) Schreiben Sie eine Hilfsfunktion `print` in Java, welche eine übergebene Matrix am Bildschirm ausgibt.

c) Schreiben Sie eine Funktion `multiply`, welche zwei übergebene Matrizen miteinander multipliziert und das Ergebnis zurückgibt.

d) Überlegen Sie sich, wie die Matrix-Multiplikation mit Hilfe von Threads gelöst werden kann und implementieren Sie diese.