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

SS 2006
Übungsblatt 4
16. Juni 2006

Übungen zu Einführung in die Informatik II

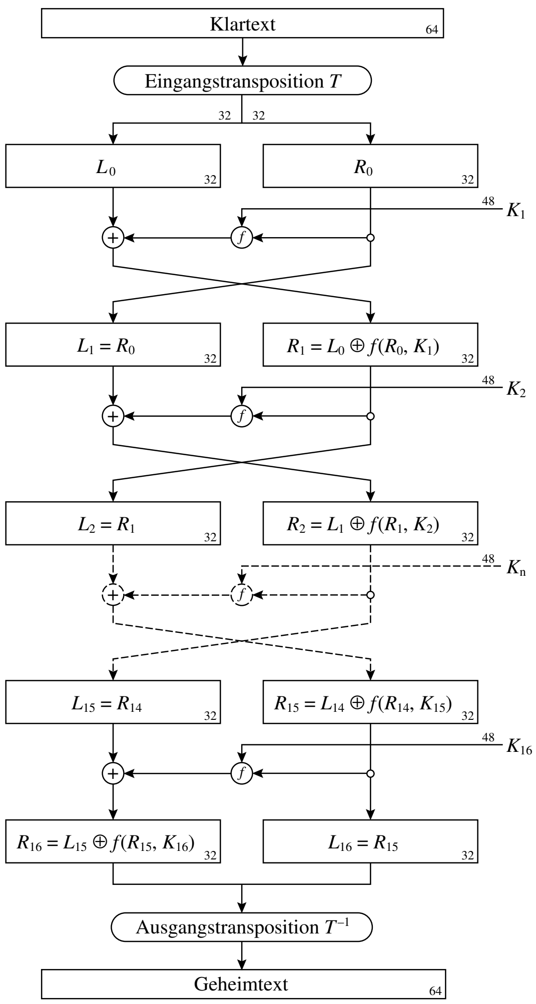
Aufgabe 12 DES Verfahren (Lösungsvorschlag)

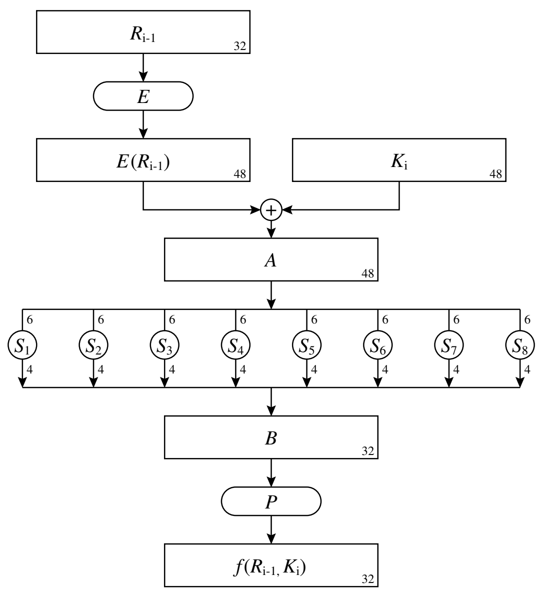
Beim DES-Verfahren wird ein Nachrichtenblock von 64 Bit Länge mit Hilfe eines Schlüssels von 56 Bit Länge chiffriert. Nach einer Eingangstransposition T wird der Klartext aufgeteilt in zwei Hälften L_0 und R_0 von je 32 Bit Länge. Darauf folgen 16 'Runden' mit

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

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.





Beweis: Die einzelnen Runden des Chiffrierens können durch die involutorischen Abbildungen

$$\begin{aligned} h_i : (R,L) &\mapsto \quad (R, L \oplus f(R,K_i)) \qquad (f - Funktion) \\ g : (R,L) &\mapsto \quad (L,R) \qquad\qquad\qquad (Vertauschen) \end{aligned}$$

beschrieben werden. Bei g ist die Involution offensichtlich, bei h_i folgt sie aus der Beziehung

$$L \oplus f(R,K_i) \oplus f(R,K_i) = L$$

Das Chiffrieren führt also insgesamt auf die Abbildung

$$DES = T^{-1} \circ h_{16} \circ g \circ h_{15} \circ g \circ \cdots \circ h_2 \circ g \circ h_1 \circ T$$

(bei der letzten Runde wird nicht mehr vertauscht). Beim Dechiffrieren wird lediglich die Reihenfolge der Teilschlüssel umgekehrt:

$$DES^{-1} = T^{-1} \circ h_1 \circ g \circ h_2 \circ g \circ \cdots \circ h_{15} \circ g \circ h_{16} \circ T$$

Die Komposition von DES und DES^{-1} ergibt wegen der Involution der einzelnen Abbildungen die identische Abbildung.

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

Auch bei dieser Aufgabe hilft es, das ganze Problem schrittweise zu betrachten. Die Eingangstransposition T ist kryptologisch irrelevant, da sie am Ende wieder aufgehoben wird. Die invertierte Nachricht $\overline{M}$ wird aufgeteilt in zwei Teile $\overline{L_0}$ und $\overline{R_0}$, wobei letzteres als Eingabe für die erste Anwendung der f-Funktion dient.

Betrachten wir die f-Funktion genauer: $\overline{R_{i-1}}$ wird zuerst expandiert durch Bitverdopplungen; dabei bleibt die Invertiertheit sicher erhalten, damit erhalten wir

$$E(\overline{R_{i-1}}) = \overline{E(R_{i-1})}$$

Dies wird per $\oplus$ (XOR bzw. Addition modulo 2) mit dem invertierten Schlüssel $\overline{K_i}$ verknüpft; die Schlüssel K_i sind bestimmte Bitstellen des Eingangsschlüssels K, d.h aus $\overline{K}$ folgt $\overline{K_i}$).

Aus der Eigenschaft

$$a \oplus b = \overline{a} \oplus \overline{b}$$

erhalten wir damit als Eingang für die "S-Funktionen" die gleiche Eingabe wie im nicht invertierten Fall:

$$A = \overline{E(R_{i-1})} \oplus \overline{K_i} = E(R_{i-1}) \oplus K_i$$

und damit auch die gleiche Ausgabe B, welche nun wieder permutiert wird und zurückgegeben.

Wir haben also

$$f(\overline{R_{i-1}}, \overline{K_i}) = f(R_{i-1}, K_i)$$

und durch die Addition modulo 2 (XOR) mit L_{i-1} ergibt sich

$$\overline{L_{i-1}} \oplus f(R_{i-1}, K_i) = \overline{R_i}$$

und $\overline{R_{i-1}}$ wird einfach kopiert, also $\overline{L_i}$. Zusammengenommen gilt also, dass nach jeder Runde die Ausgaben wieder invertiert sind – da das letztendliche Ergebnis (bis auf die Ausgangstransposition T^{-1}) genau die Ausgabe der letzen Runde ist, also $\overline{R_{16}} \circ \overline{L_{16}}$, haben wir also q.e.d.

$$DES_{\overline{K}}(\overline{M}) = \overline{DES_K(M)}$$

c) Ausführung per Hand: Versuchen Sie sich zunächst den Ablauf des Verfahrens klar zu machen indem Sie die ersten Schritte mit Papier und Bleistift durchführen. Für die weiteren Schritte können Sie das Programm *Des-verbose.java* benutzen, das den Algorithmus Schritt für Schritt ausführt und jeweils die wichtigsten Zwischenergebnisse ausgibt.

d) Eine mögliche Implementierung könnte wie folgt aussehen (das abgedruckte Programm kompiliert nicht, da alle Tabellen entfernt wurden, benutzen Sie stattdessen *Des-verbose.java* oder *Des.java*):

```
import java.lang.String;
import java.lang.Integer;

class Des{
  static public void main(String args[]) {

    String mm = "0123456789abcdef";
    long xx = Long.parseLong(mm, 16);
    String binXX = Long.toBinaryString(xx);

    boolean[] text = new boolean[64];
    boolean[] pkey = new boolean[64];
    for (int i = 0; i < 64; i++) {
      text[i] = false;
      pkey[i] = false;
    }

    // Klartext in Binaerdarstellung umwandeln
    for (int i = 0; i < binXX.length(); i++) {
      if (binXX.charAt(binXX.length() - (i+1)) == '1') text[63 - i] = true;
    }

    String kk = "fedcba9876543210";
    xx = Long.parseLong(kk, 16);
    binXX = Long.toBinaryString(xx);

    // Schluessel in Binaerdarstellung umwandeln
    for (int i = 0; i < binXX.length(); i++) {
      if (binXX.charAt(binXX.length() - (i+1)) == '1') pkey[63 - i] = true;
    }

    // Verschluesseln
    boolean[] cifer = new boolean[64];
    des(text, pkey, cifer, true);

    // Wieder entschluesseln
    boolean[] original = new boolean[64];
    des(cifer, pkey, original, false);
  }

  static void des(boolean[] text, boolean[] pkey, boolean[] cifer, boolean encode) {

    int[] T, T_I, keyTrans, keyComp, RTrans, BTrans = { ... };
    int[][][] S  = {{{ ... }, ... }, ... };
```

```
boolean[] L, R, skey, rkey, Rexp, A, B, PB, result, pertext, perkey = new boolean[N];

// Eingangspermutation
for (int i = 0; i < 64; i++) {
  pertext[i] = text[63 - T[i]];
}

// Schluesselpermutation
for (int i = 0; i < 56; i++) {
  perkey[i] = pkey[63 - keyTrans[i]];
}

for (int i = 0; i < 32; i++) {
  L[i] = pertext[i];
  R[i] = pertext[i + 32];
}

// Jetzt kommen die 16 Runden
for (int i = 0; i < 16; i++) {

  int shift;
  if (encode) shift = (i+1) / 2 + ((i+2) / 2) * 2;
  else {
    int u = 15 - i;
    shift = (u+1) / 2 + ((u+2) / 2) * 2;
  }

  // zyklische Verschiebung des Schluessels
  for (int j = 0; j < 56; j++) {
    if (j + shift < 28) skey[j] = perkey[j + shift];
    if ((j + shift >= 28) && (j < 28)) skey[j] = perkey[j + shift - 28];
    if ((j + shift < 56) && (j >= 28)) skey[j] = perkey[j + shift];
    if (j + shift >= 56) skey[j] = perkey[j + shift - 28];
  }

  // Erzeugung des Rundenschluessels
  for (int j = 0; j < 48; j++) {
    rkey[j] = skey[55 - keyComp[j]];
  }

  // Expansion der rechten Seite
  for (int j = 0; j < 48; j++) {
    Rexp[j] = R[31 - RTrans[j]];
  }

  // XOR Verknuepfung der Ergebnisse
  for (int j = 0; j < 48; j++) {
    A[j] = rkey[j] ^ Rexp[j];
  }

  // Selektoren fuer S-Box berechnen
  int[] x = new int[8];
  int[] y = new int[8];
  int[] boxEntry = new int[8];
```

```
      for (int j = 0; j < 8; j++) {
        if (A[j * 6])     x[j] += 2;
        if (A[j * 6 + 5]) x[j] += 1;
        if (A[j * 6 + 1]) y[j] += 8;
        if (A[j * 6 + 2]) y[j] += 4;
        if (A[j * 6 + 3]) y[j] += 2;
        if (A[j * 6 + 4]) y[j] += 1;
        boxEntry[j] = S[j][x[j]][y[j]];
      }

      // S-Box-ERgebnis in Binaerdarstellung umwandeln
      for (int j = 0; j < 8; j++) {
        for (int k = 0; k < 4; k++) {
          if (boxEntry[j] % 2 == 1) B[j * 4 + (3 - k)] = true;
          else B[j * 4 + (3 - k)] = false;
          boxEntry[j] /= 2;
        }
      }

      // Permutation zum Abschluss von f
      for (int j = 0; j < 32; j++) {
        PB[j] = B[31 - BTrans[j]];
      }

      // XOR mit linker Seite -> neue rechte Seite
      if (i < 15) {
        for (int j = 0; j < 32; j++) {
          tmp[j] = R[j];
          R[j] = L[j] ^ PB[j];
          L[j] = tmp[j];
        }
      }
      // letzte Runde: XOR mit linker Seite -> neue linke Seite
      else {
        for (int j = 0; j < 32; j++) {
          L[j] = L[j] ^ PB[j];
        }
      }
    }

    // Ergebnis zusammenfassen
    for (int i = 0; i < 32; i++) {
      result[i] = L[i];
      result[i + 32] = R[i];
    }

    // Eingangspermutation rueckgaengig machen
    for (int i = 0; i < 64; i++) {
      cifer[i] = result[63 - T_I[i]];
    }
  }
}
```

Aufgabe 13 Threads in Java (Lösungsvorschlag)

a) PrintThread.java

```
import java.lang.Thread;

public class PrintThread extends Thread {
  private String text;

  PrintThread(String text) {
    this.text = text;
  }

  public void run() {
    while (true) {
      System.out.println(text);
    }
  }
}
```

Main.java

```
public class Main {
  public static void main(String[] args) {
    new PrintThread("tick").start();
    new PrintThread("tack").start();
    new PrintThread("tock").start();
  }
}
```

b) PrintThreadYield.java

```
import java.lang.Thread;

public class PrintThreadYield extends Thread {
  private String text;

  PrintThreadYield(String text) {
    this.text = text;
  }

  public void run() {
    while (true) {
      System.out.println(text);
      yield();
    }
  }
}
```

c) PrintThreadSleep.java

```
import java.lang.Thread;
```

```java
public class PrintThreadSleep extends Thread {
  private String text;

  PrintThreadSleep(String text) {
    this.text = text;
  }

  public void run() {
    while (true) {
      System.out.println(text);

      try {
        sleep(500);
      } catch (java.lang.InterruptedException e) {
        break;
      }
    }
  }
}
```

Aufgabe 14 Matrix-Multiplikation mit Threads in Java (Lösungsvorschlag)

- $$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{pmatrix} = \begin{pmatrix} 14 & 14 \\ 14 & 14 \end{pmatrix}$$

- Main.java

```java
public class Main {
  public static void print(float[][] matrix) {
    for (int i = 0; i < matrix.length; i++) {
      for (int j = 0; j < matrix[0].length; j++) {
        System.out.print(matrix[i][j] + "\t");
      }

      System.out.println();
    }
  }

  public static void main(String[] args) {
    float[][] a = {
      {1.0f, 2.0f, 3.0f},
      {1.0f, 2.0f, 3.0f}
    };

    print(a);
    System.out.println();

    float[][] b = {
```

```
      {1.0f, 1.0f},
      {2.0f, 2.0f},
      {3.0f, 3.0f}
    };

    print(b);
    System.out.println();

    MatrixMultiply mm = new MatrixMultiply(a, b);

    mm.multiply();
    print(mm.c);
    System.out.println();

    MatrixMultiply mm2 = new MatrixMultiply(a, b);

    mm2.multiply2();
    print(mm2.c);
  }
}
```

- MatrixMultiply.java

```
public class MatrixMultiply {
  public float[][] a;
  public float[][] b;
  public float[][] c;

  MatrixMultiply(float[][] a, float[][] b) {
    this.a = a;
    this.b = b;
    this.c = new float[a.length][b[0].length];
  }

  public void multiply() {
    for (int i = 0; i < a.length; i++) {
      for (int j = 0; j < b[0].length; j++) {
        for (int k = 0; k < a[0].length; k++) {
          c[i][j] += a[i][k] * b[k][j];
        }
      }
    }
  }

  public void multiply2() {
    MatrixMultiplyThread[] threads = new
        MatrixMultiplyThread[a.length];

    for (int i = 0; i < a.length; i++) {
      threads[i] = new MatrixMultiplyThread(i, this);
      threads[i].start();
    }
```

```
    for (int i = 0; i < a.length; i++) {
      try {
        threads[i].join();
      } catch (java.lang.InterruptedException e) {
        break;
      }
    }
  }
}
```

- MatrixMultiplyThread.java

```
import java.lang.Thread;

public class MatrixMultiplyThread extends Thread {
  private int i;
  private MatrixMultiply mm;

  MatrixMultiplyThread(int i, MatrixMultiply mm) {
    this.i = i;
    this.mm = mm;
  }

  public void run() {
    for (int j = 0; j < mm.b[0].length; j++) {
      for (int k = 0; k < mm.a[0].length; k++) {
        mm.c[i][j] += mm.a[i][k] * mm.b[k][j];
        yield();
      }
    }
  }
}
```