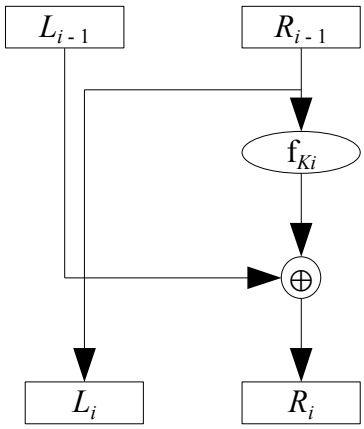
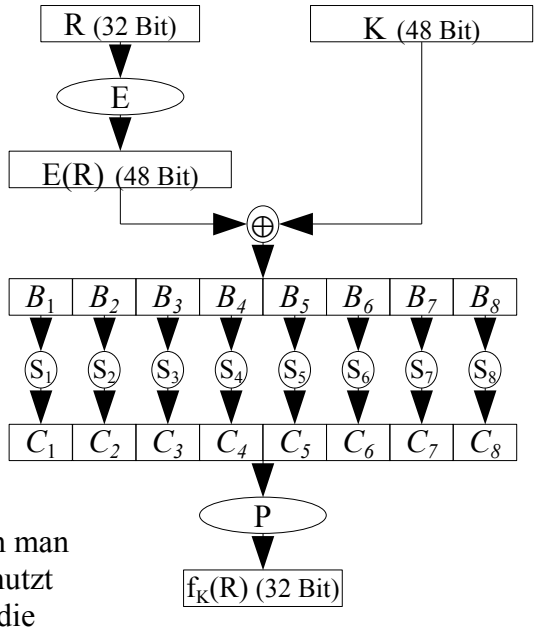
DES (Data Encryption Standard)



- DES ist ein Feistel-Chiffre:
  - Blockchiffre, symmetrisch
  - Rundenbasiert (r Runden)
  - Rundenschlüssel: R_i $(1 \leq i \leq r)$
  - Verschlüsselungsfunktion f
  - Klartext p =: (L_0 , R_0) der Länge 2t (L_0, R_0 je t)
  - Für i = 1, ..., r: $(L_i , R_i) := (R_{i-1} , L_{i-1} \oplus f_{Ki}(R_{i-1}))$
  - Schlüsseltext c := (L_r , R_r) [nach r Runden]
  - Entschlüsselung: $(R_{i-1} , L_{i-1}) := (L_i , R_i \oplus f_{Ki}(L_i))$
    Rückgewinnung des Klartextpaares (R_0 , L_0) durch Verwenden der Rundenschlüssel in inverser Reihenfolge
- DES hat Rundenschlüssellänge 48 Bit
- Rundenzahl r = 16
- Blocklänge 64 Bit
- Schlüssellänge: 64 Bit (davon jedes 8. Bit redundant) ⟶ effektive Schlüssellänge 56 Bit
- Permutation des Klartextes vor der ersten Runde mit fester Permutation IP, Permutation des Schlüsseltextes nach der letzten Runde mit IP^{-1}
- Schlüsselexpansion:
  - Definiere Schrittweitefunktion vi := 1, falls $i \in \{1,2,9,16\}$, sonst 2
  - Feste Extraktionsfunktion: $PC1 : \{0,1\}^{64} \rightarrow \{0,1\}^{28} \times \{0,1\}^{28}$
  - Feste Auswahlfunktion: $PC2 : \{0,1\}^{28} \times \{0,1\}^{28} \rightarrow \{0,1\}^{48}$
  - Setze (C_0 , D_0) := PC1(k) [hier ist k der Schlüssel der Länge 64 Bit]
    PC1 entfernt die Redundanzen aus k und permutiert das Ergebnis
  - Gewinne induktiv C_i aus C_{i-1} durch zyklischen Linksshift um vi Stellen (D_i aus D_{i-1} ebenso)
  - Setze für i = 1, ..., r die Rundenschlüssel K_i := PC2(C_i , D_i)
- innere Verschlüsselungsfunktion $f_{Ki}(R)$:
  - Expansionsfunktion E: Vergrößert die Eingabe R durch Verdopplung mancher Bits + Permutation
  - Exklusives Verodern des Rundenschlüssels K mit E(R): E(R) ⊕ K
  - Aufteilen des Zwischenergebnis in 8 Blöcke zu je 6 Bit: B_i
  - Anwenden einer festen monoalphabethischen Substitution (S-Boxen) auf B_i:
    $C_i := S_i(B_i)$ [C_i hat Länge 4 Bit]
  - Permutation der Konkatenation aller C_i mit fester Permutationsfunktion P
  - Das Ergebnis dieser Permutation ist das Ergebnis der inneren Verschlüsselung



- Entschlüsselung: DES lässt sich entschlüsseln, indem man die Rundenschlüssel in umgekehrter Reihenfolge benutzt
- Es gibt 4 schwache und 12 halbschwache Schlüssel, die explizit vermieden werden sollten
- Gute Sicherheit gegen Kryptoanalyse, aber zu kurze Schlüssellänge (56 Bit!)
- Größere Schlüssellänge durch Kaskade: $3DES(K1,K1,K3,p) := DES_{K3}(DES^{-1}_{K2}(DES_{K1}(p)))$

AES (Advanced Encryption Standard)

- Spezialisierung des symmetrischen Blockchiffre „Rijndael“
- byte := 8 Bit, word := 4 byte = 32 Bit
- Blocklänge 4 words = 128 Bit
- Schlüssellänge: Nk words = NK * 4 bytes (Nk = 4,6 oder 8)
- Rundenlänge Nr: abhängig von Nk (Nk = 4 ⇒ Nr = 10; Nk = 6 ⇒ Nr = 12; Nk = 8 ⇒ Nr = 14)
- „state“: 4x4 Byte-Matrix. Jede Spalte entspricht einem word (von oben nach unten)
- Klartext p hat Darstellung als state
- Expandierter Schlüssel w: Byte-Matrix von 4x(4*(Nr+1)), enthält die (Nr+1) Rundenschlüssel
- Verschlüsselunsalgorithmus:

```
Cipher(byte[4,Nb] in, word[Nk] k) returns byte[4,Nb] {
  byte[4,Nb] state = in;
  word[Nb*(Nr+1)] w = KeyExpansion(k);
  state = AddRoundKey(state, w[0 to Nb-1]);
  for (round = 1 to Nr-1) {
    state = SubBytes(state);
    state = ShiftRows(state);
    state = MixColumns(state);
    state = AddRoundKey(state, w[round*Nb to (round+1)*Nb-1]);
  }
  state = SubBytes(state);
  state = ShiftRows(state);
  state = AddRoundKey(state, w[Nr*Nb to (Nr+1)*Nb-1]);
  return state;
}
```

  - Interpretation von Bytes als Elemente des Körpers $GF(2^8)$ mit:
    - Addition: Bitweise XOR
    - Multiplikation: Binäre Multiplikation modulo '100011011' =: m
    - Inverses: zu $b \neq 0$ gibt es b^{-1} mit $b^{-1} * b = 00000001 \bmod m$. ($0^{-1} := 0$)
  - AddRoundKey: byteweise externe Veroderung von state und Rundenschlüssel
  - SubBytes: Es gibt eine feste, invertierbare Matrix A und einen festen Bitvektor c. Definiere subst(b) := $A*b^{-1} \oplus c$ für Bytes b. SubBytes wendet diese „S-Box“ auf jedes Byte von state an. Da b nur 256 Werte annehmen kann: Realisierung als Tabelle
  - ShiftRows: Zyklischer Linksshift der Zeilen (1. Zeile: gar nicht, 2. Zeile: um 1 Byte, 3.Zeile: um 2 Byte, 4. Zeile: um 3 Byte)
  - MixColumns: Diffusion der Spalten von state: Für jede Spalte mit Polynomdarstellung $s_j = s_{0,j} + s_{0,j}\, x + s_{0,j}\, x^2 + s_{0,j}\, x^3$ erhalte: $s_j' := (s_j * a(x)) \bmod (x^4 + 1)$ mit
    $a(X) := '03'\, x^3 + '01'\, x^2 + '01'\, x + '02'$ ['xy': hexadezimale Darstellung des Bytes]
- Schlüsselexpansion:
  - Die ersten Nk Spalten von w werden mit k gefüllt
  - Für alle weiteren i-ten Spalten von w:
    - Temp := (i-1)te Spalte von w = w[i-1]
    - Falls i Vielfaches von Nk:
      temp = SubWord(RotWord(temp)) ⊕ rcon[i/Nk]
    - Falls Nk = 8 und n modulo 8 = 4:
      temp = SubWord(temp)
    - Die i-te Spalte von w ist: w[i] = temp ⊕ w[i-Nk]
  - SubWord: Wendet auf jedes Byte des words die S-Box *subst* an
  - RotWord: Zyklischer Linksshift des words um 1 Byte
  - rcon[i]: Byte-Vektor (=word): $('02'^{(i-1)}, '00', '00', '00')$
    Hierbei ist die Potenzierung von '02' als Multiplikation auf $GF(2^8)$ zu vestehen

- Entschlüsselungsalgorithmus:

```
DeCipher(byte[4,Nb] in, word[Nk] k) returns byte[4,Nb] {
  byte[4,Nb] state = in;
  word[Nb*(Nr+1)] w = KeyExpansion(k);
  state = AddRoundKey(state, w[Nr*Nb to (Nr+1)*Nb-1]);
  for (round = Nr-1 downto 1) {
    state = InvShiftRows(state);
    state = InvSubBytes(state);
    state = AddRoundKey(state, w[round*Nb to (round+1)*Nb-1]);
    state = InvMixColumns(state);
  }
  state = InvShiftRows(state);
  state = InvSubBytes(state);
  state = AddRoundKey(state, w[0 to Nb-1]);
  return state;
}
```

  - Rückwärtsausführen aller Schritte: (daher leichte Variation der Reihenfolge)
  - AddRoundKey ist zu sich selbst invers (Reihenfolge der Rundenschlüssel invertieren!)
  - InvShiftRows: Wie ShiftRows, nur Shift nach rechts um den gleichen Betrag
  - InvMixColumns: Wie MixColumns, nur mit inversem Polynom:
    $a^{-1}(X) := \text{'0B'}\ x^3 + \text{'0D'}\ x^2 + \text{'09'}\ x + \text{'0E'}$
  - InvSubBytes: Zu subst gibt es inverse Substitution $subst^{-1}$ (Substitutionstabellen!)
- Sicherheit:
  - AES gehört zu den sichersten (und performantesten) symmetrischen Blockchiffren
  - Robustheit gegen statistische Angriffe und lineare/differentielle Kryptoanalyse wegen nichtlinearität und starker Konfusion und Diffusion
  - nur theoretische Angriffe bekannt, deren Rechenaufwand utopisch ist (ab 2^{100} Rechenschritte)