SECOM: Sichere Online Verschlüsselung für ISDN-Geräte

Mathias Schmalisch · Hagen Ploog · Dirk Timmermann

Universität Rostock

Übersicht

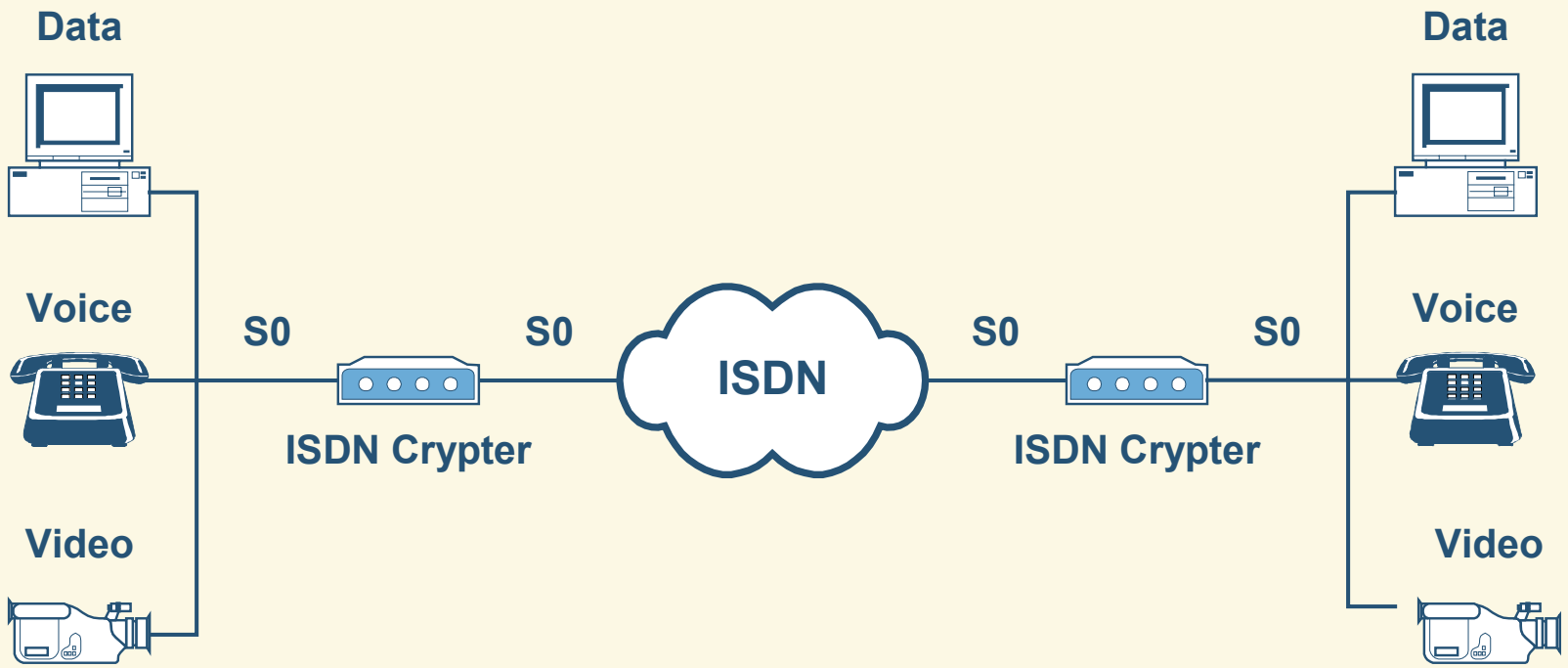
- Laufende Arbeiten
- Motivation
- Kryptographie
- Implementierung
- Zusammenfassung

Laufende Arbeiten

- Ziele:
  - Hochgeschwindigkeitsverschlüsselung für multimediale Datenströme
  - Kleine Architekturen für den Bereich Smartcard
- Weg:
  - Optimierung von Algorithmen für den Hardwareeinsatz
  - Einsatz neuer Architekturen und Schaltungstechniken
- Projekte:
  - SECOM mit Triple-DES für Daten und RSA für Schlüsselaustausch
  - Online Festplattenverschlüsselung
  - TDES: GHz Kryptoprozessor durch massives Pipelining und dynamischer Schaltungstechnik (TSPC)
  - Smartcard RSA: Minimallösungen für modulare Exponentiation

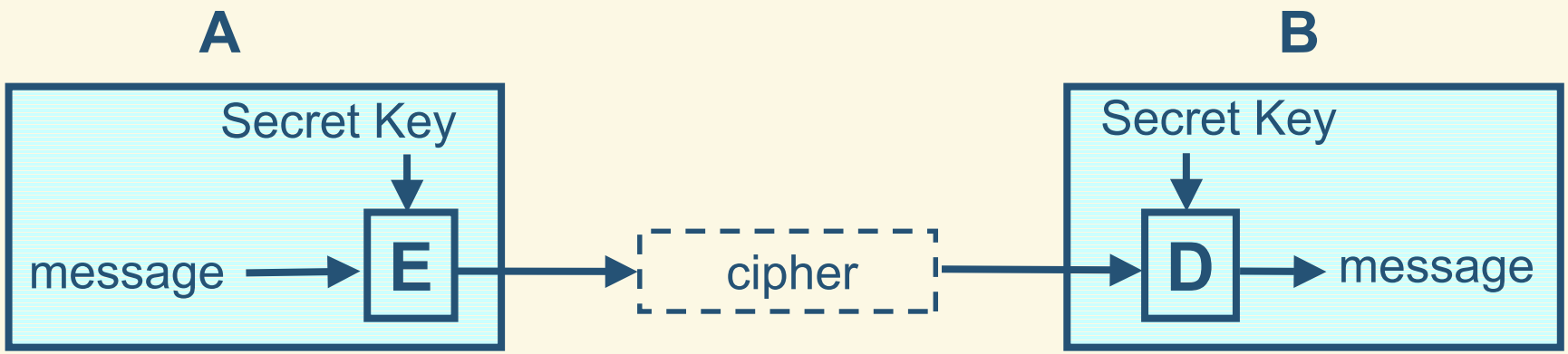
Motivation

- Benutzerfreundlich und transparent zu gängigen ISDN-Geräten
- Wiederverwendung von vorhandener Hardware



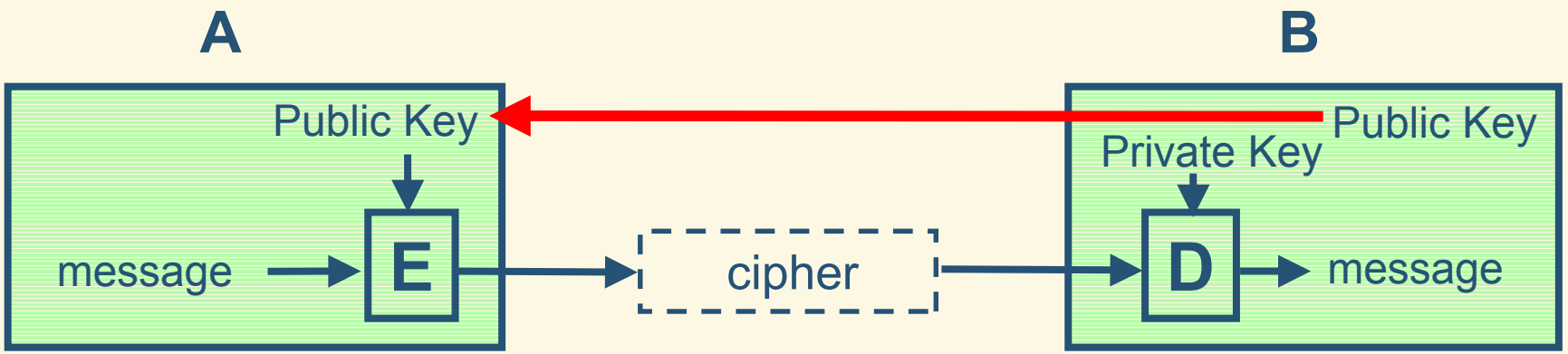
Einführung: Symmetrische Systeme

- Ein geheimer Schlüssel auf beiden Seiten
  - Verschlüsselung : $c = f(m, k)$
  - Entschlüsselung : $m = f^{-1}(c, k)$



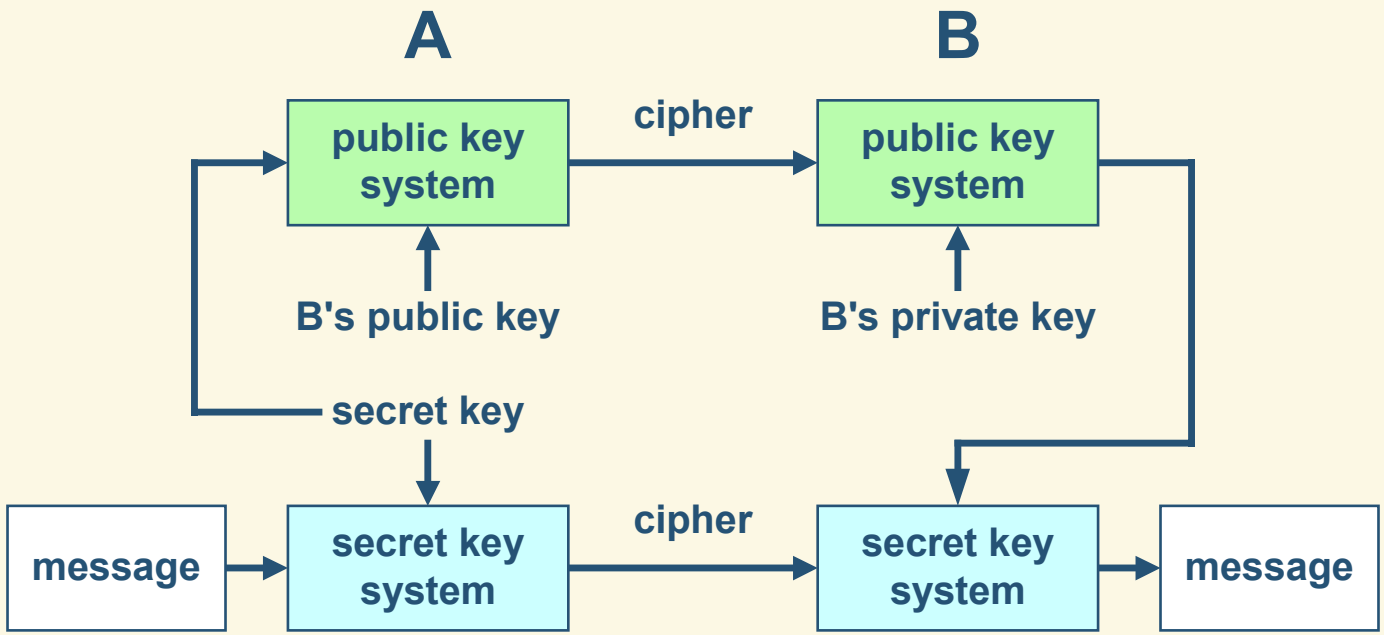
Einführung: Asymmetrische Systeme

- Verschiedene Schlüssel auf jeder Seite
  - Verschlüsselung : $c = f(m, \text{public key})$
  - Entschlüsselung : $m = f^{-1}(c, \text{private key})$
- Langsamer als symmetrische Systeme (~ 100 mal)



Einführung: Hybride Systeme

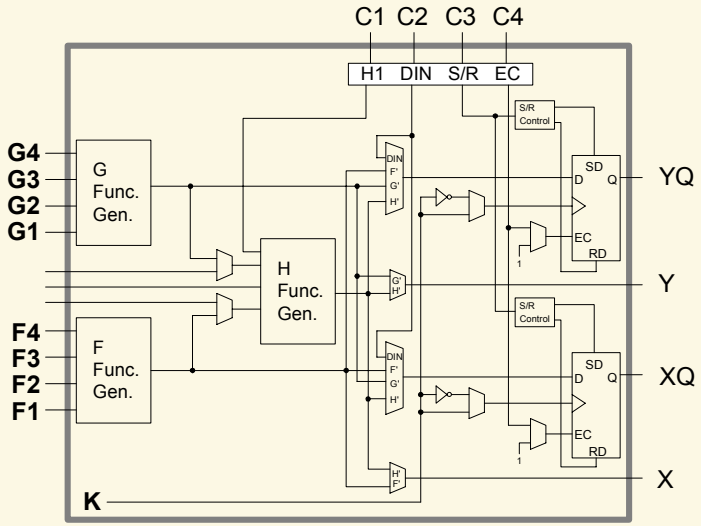
- Benutzung des asym. Systems für den Sitzungsschlüssel
  - Während des Verbindungsaufbaus
  - Unkritische Geschwindigkeitsanforderung
- Benutzung des sym. Systems für den Datenstrom

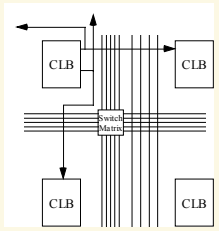


Einführung: FPGA Xilinx XC4000

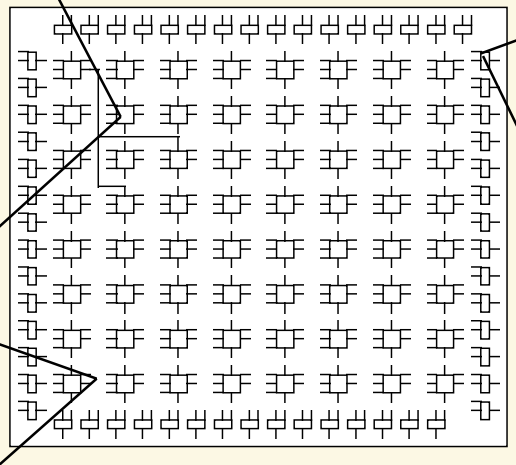
- **Große Dichte → 1 Mio. System Gatter**
- **SRAM basierende LUT**
- **Array Struktur**
- **interne Tri-States**
- **beliebig rekonfigurierbar in wenigen ms**

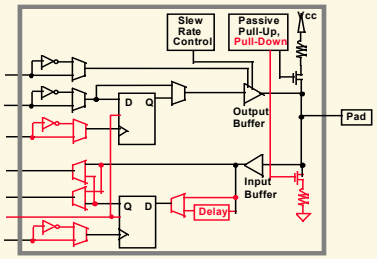
Configurable Logic Blocks (CLBs)





Programmable Interconnect

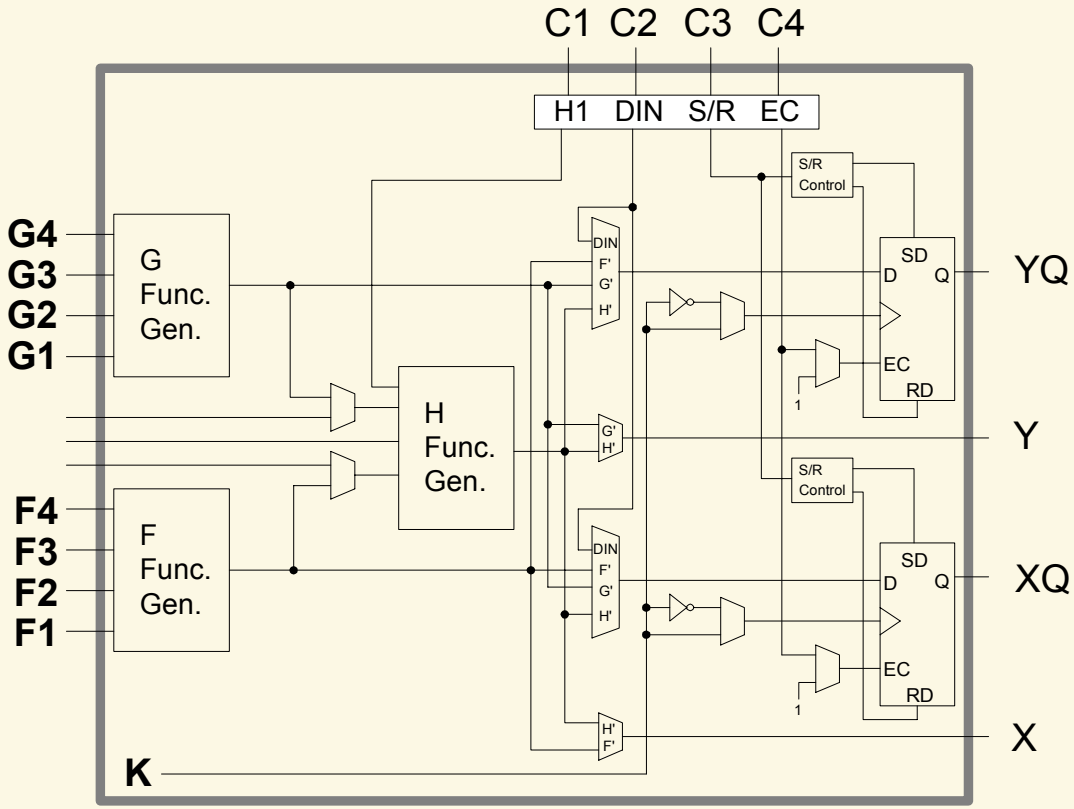




I/O Blocks (IOBs)

Einführung: CLB

- 2 Funktionsgeneratoren (LUT)
  - 16x1 RAM o. ROM / Logik
- 2 Register
  - FlipFlop / Latch
  - freie Flankenauswahl
  - RESET
    - Synchron / asynchron



Asymmetrisches System: RSA

- RSA : Modulare Exponentiation
  - Verschlüsselung : $E(m) = m^{E} \bmod N = c$ • public key : E, N
  - Entschlüsselung : $D(c) = c^{D} \bmod N = m$ • private key : D

Sicherheit ist abhängig von der Länge der Zahl N (> 512 bits)

Realisation: "Square and multiply" (Knuth): $Y = B^{E} \bmod N$

```
Y = 1
FOR i = log2(E) DOWNTO 0 DO
    Y = Y * Y mod N
    Y = Y * B mod N if (Ei == 1)
END
```

=> Exponentiation reduziert auf ~ 1.5 n modulare Multiplikationen, $n = \log_2(N)$

Multiple precision

- Zwischenergebnisse müssen in externen RAM der Breite w gespeichert werden
  - Neue Basis $W=2^w$ anstelle der Basis 2
  - Transformiere jede Integerzahl X in die neue Basis

$$X = \sum_{i=0}^{x-1} x_i 2^i = \sum_{i=0}^{s-1} d_i (2^w)^i = \sum_{i=0}^{s-1} d_i W^i$$

- Herz des Algorithmus ist eine $w \bullet w$ Bit Multiplikation
  - z Anzahl Takte für die Ausführung einer $w \bullet w$ Bit Multiplikation
- VHDL-Model kann parametrisiert werden (n, w, z)

Ergebnisse: RSA

Zeit für die Berechnung einer mod. Exponentiation: $$\frac{3 \times z \times (n^3 - n^2)}{f_{clk} \times w^2}$$

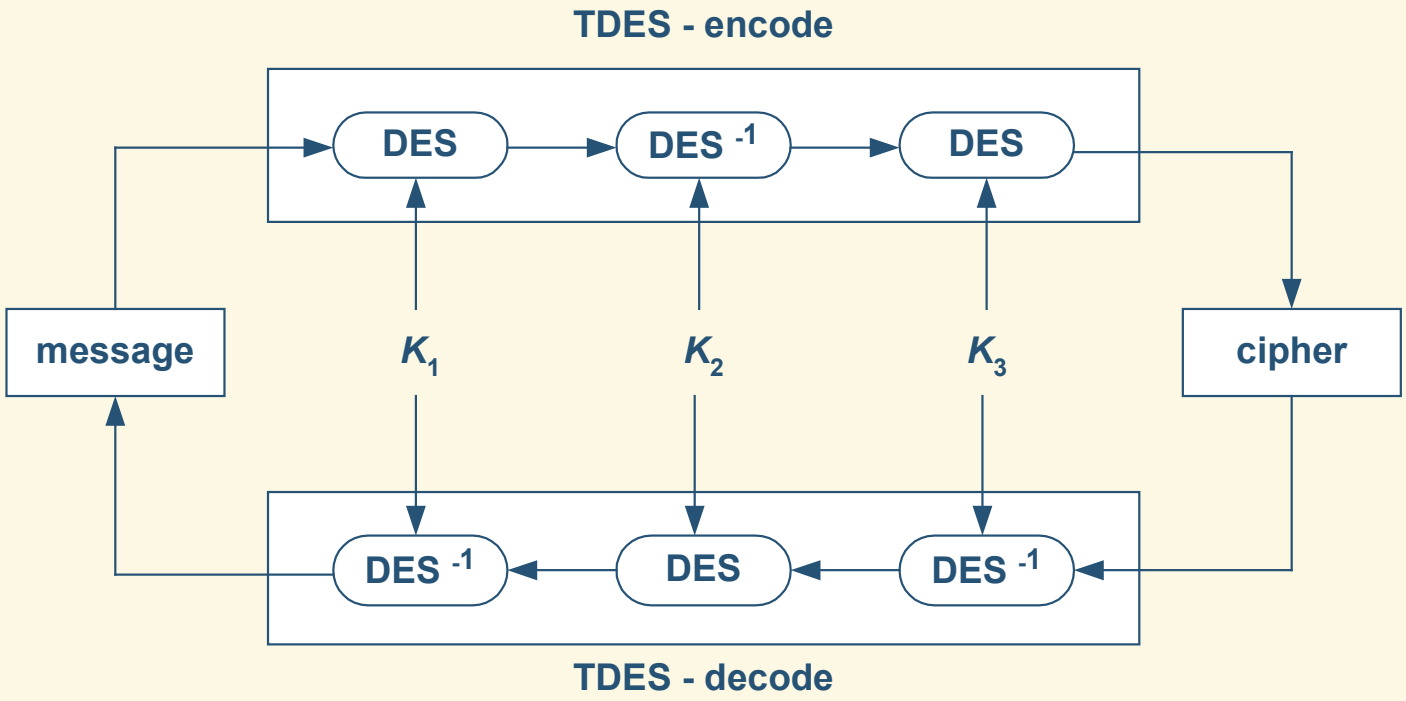
Ergebnisse [s] für 3.5 MHz

w \ n	256	512	1024
8	0.22	1.79	14.37
16	0.056	0.45	3.59
32	0.014	0.11	0.90
64	0.0035	0.028	0.22

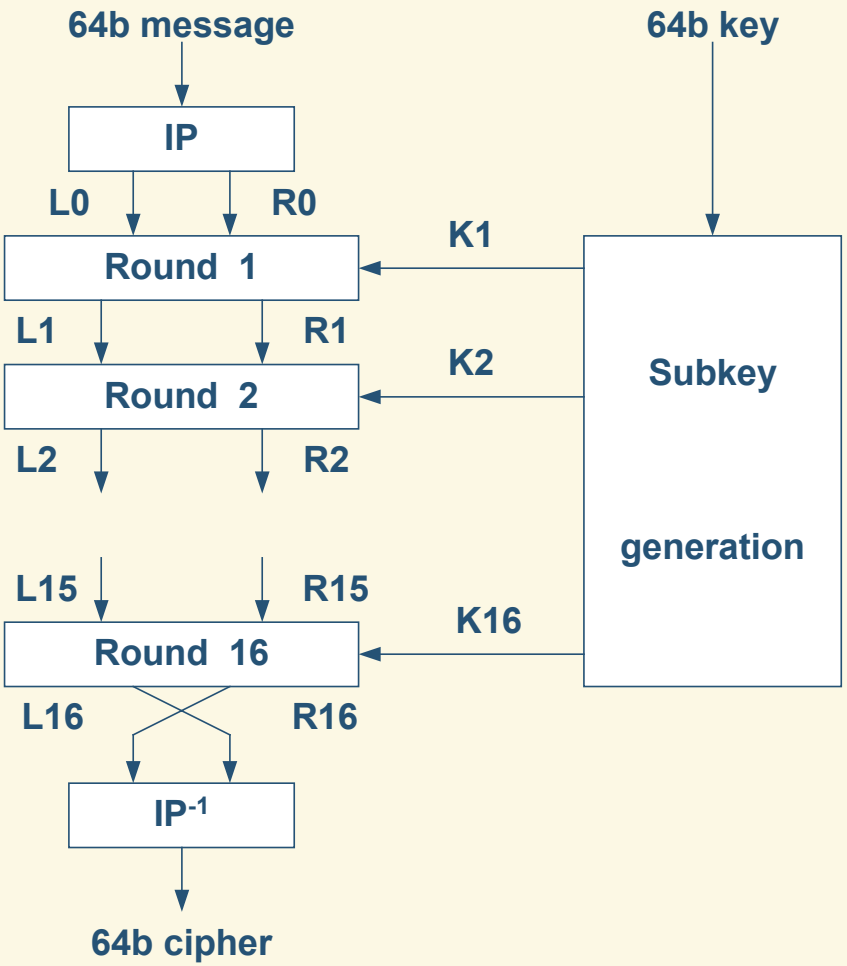
Realistische Werte

Symmetrisches System: Triple DES (TDES)

- TDES : kaskadiertes DES, zur Zeit noch nicht gebrochen



DES



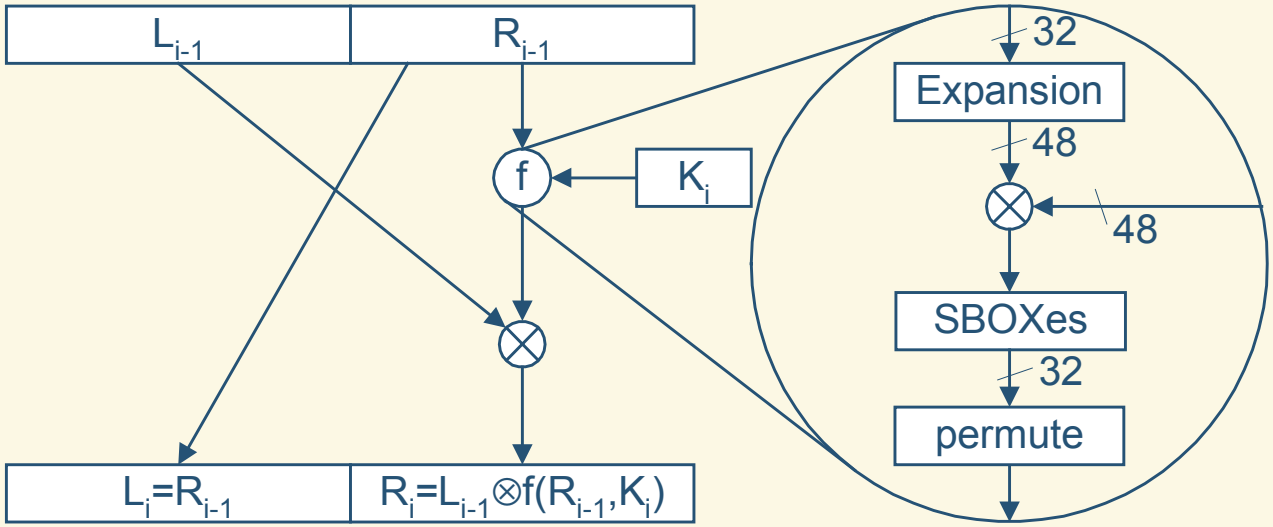
- 64 Bit Input
- 64 Bit Output
- 16 Runden

$$L_i = R_{i-1}$$

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

- Jede Runde hat eigenen Teilschlüssel K_i
- Runden sind unabhängig voneinander

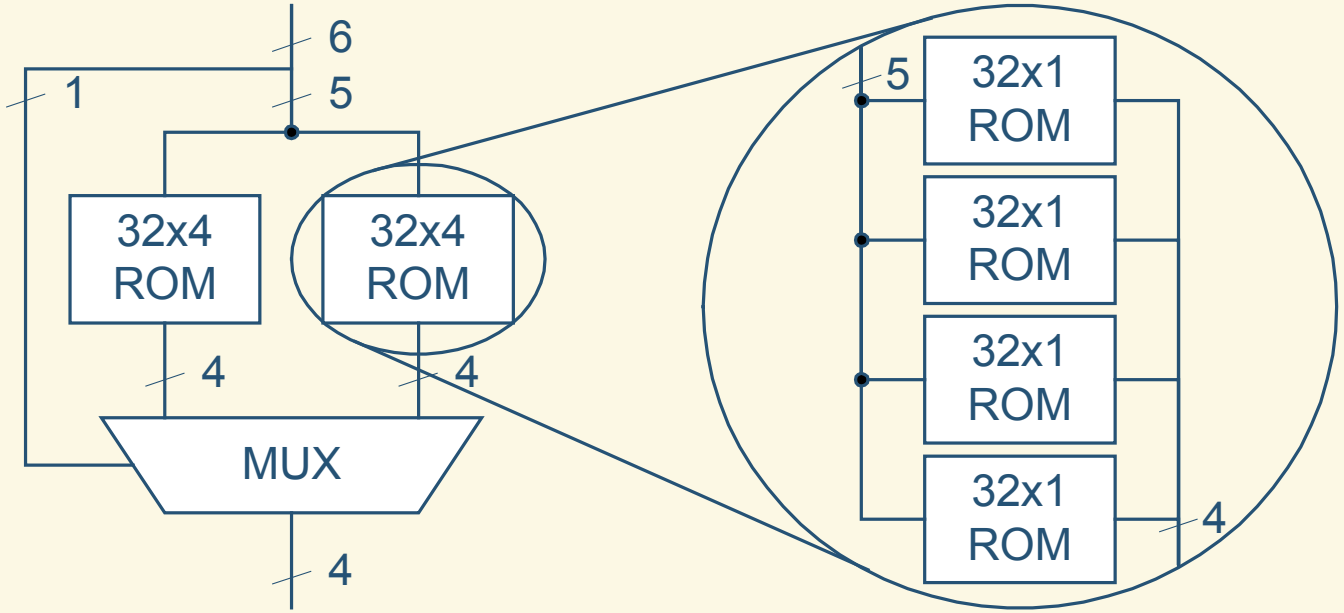
Eine Runde



- Die f-Funktion ist das kryptographische Herz beim DES
  - Erweitere R auf 48 Bit
  - XOR Ergebnis mit K_i
  - Ausführen der SBoxen (Reduzieren auf 32 Bits)
  - Permutiere Ergebnis der SBoxen

SBox

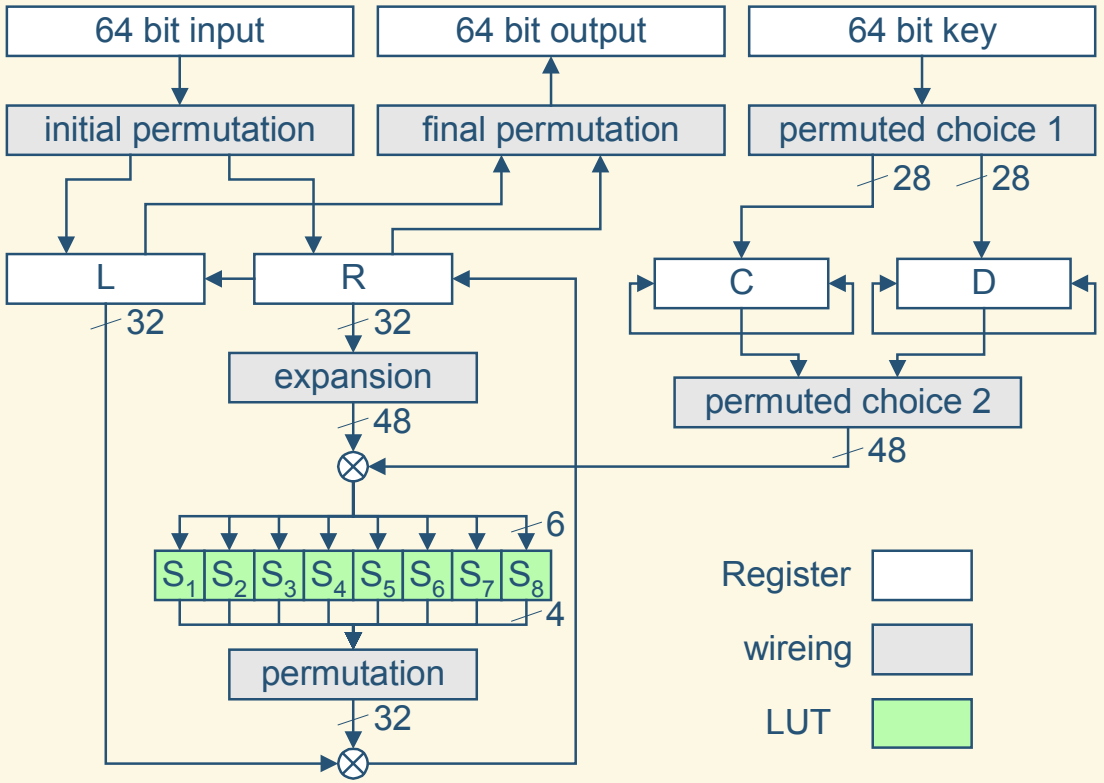
- Eine SBox läßt sich am besten mit LUT‘s realisieren



- 10 CLB’s / SBox
  - ► insgesamt 80 CLB‘s

DES: Rekursive Implementation

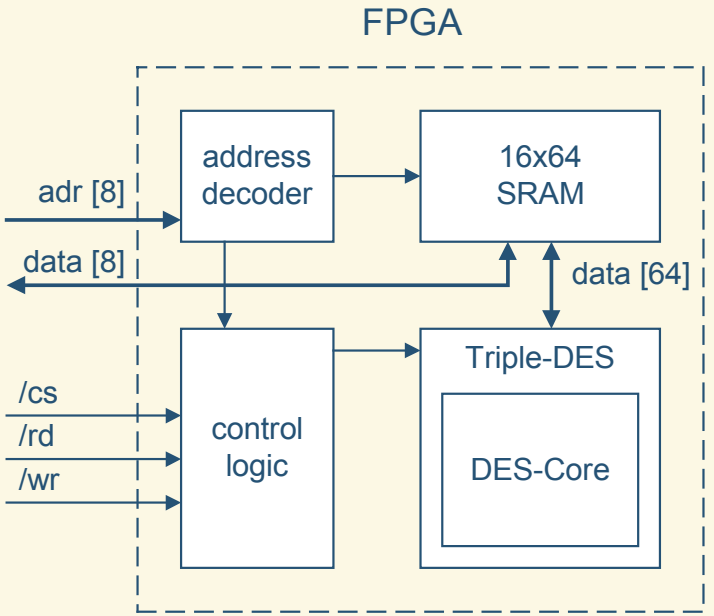
- Rekursive Implementierung für kleine Architektur



Ergebnisse: TDES

- FPGA: XC4010e-3
  - 380 CLB von 400 = 95% Auslastung
  - Aber: 209 FF von 800 = 26%
- TDES: 55 Takte
  - Datenrate @ 8 MHz: 9.3 MBits/s

ISDN B-Kanal: 64 KBit/s pro Richtung



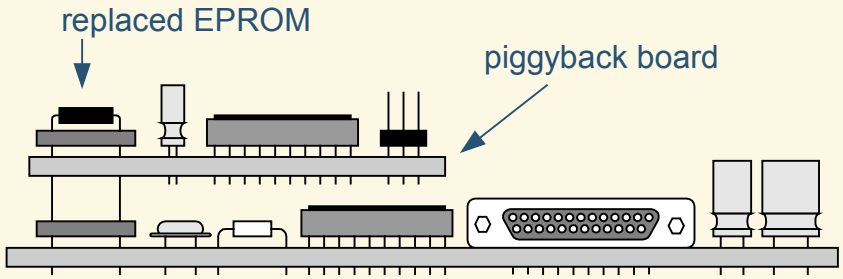
=> TDES-Core kann theoretisch 8 B-Kanäle verschlüsseln

Ausnutzung der Rekonfigurierbarkeit

- Ziel: Vorhandene Hardwareresourcen ausnutzen
- Benutzen von zeitliche getrennten Algorithmen
- Zwei Arbeitsphasen
  - RSA nach dem Neustart
  - FPGA rebooten nach erfolgreichem Schlüsselaustausch
  - Triple DES für die Verschlüsselung des Datenstroms
- Rebooten des FPGA mit neuem Inhalt in ungefähr 35 ms

Prototype

- Basiert auf einem Least-Cost-Router mit 16bit CPU
- FPGA XC4010e-3



Zusammenfassung

- ISDN Echtzeitanforderungen lasten FPGAs nicht aus
- Kryptographische Algorithmen passen in kleine FPGAs
- Rebooten des FPGAs hält die Kosten des Systems niedrig
- Verringerte zusätzliche Verdrahtung durch Huckepack-Board

Modulare Multiplikation (Montgomery)

- Schnellster bekannter Algorithmus für die modulare Multiplikation
- Berechnet das N-residue Produkt von zwei N-residue Zahlen

MontgomeryProdukt $(\overline{a}, \overline{b})$

STEP 1: $t = \overline{a} \bullet \overline{b}$

STEP 2: $u = (t + (t \bullet N' \bmod R) \bullet \mathrm{N}) / R$

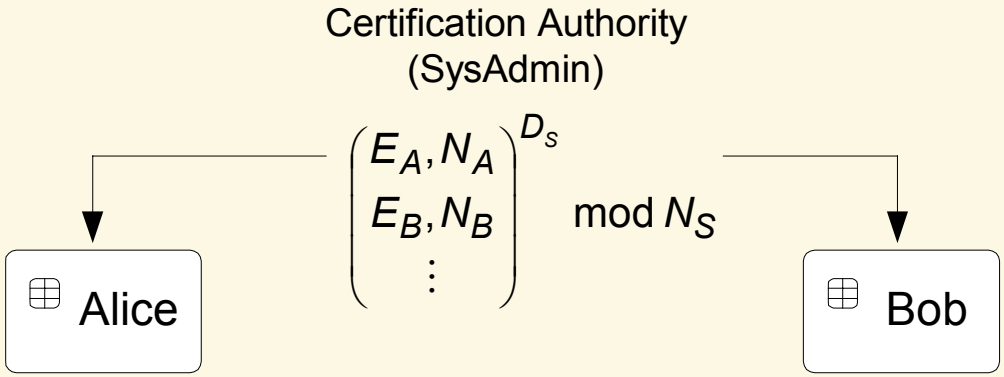
STEP 3: if $u \geq N$ then return u-N else return u

- schnell, wenn $R = 2^x$, funktioniert aber mit jedem R, welches relativ prim zu N ist
- Aber: Input und Output umsetzen in N-residue Ebene

Probleme

- Auto-detect baugleiches SECOM-Gerät
  - Hier nicht näher aufgeführt (in Software implementiert)

Probleme: Schlüsselmanagement



$c_{A1} = (key_A, IV_A)^{D_A} \bmod N_A$

$c_{A2} = (c_{A1}, N^o)^{E_B} \bmod N_B$

$c_{B1} = (key_B, IV_B)^{D_B} \bmod N_B$

$c_{B2} = (c_{B1}, N^o)^{E_A} \bmod N_A$

c_{A2} →

← c_{B2}

$m_{A2} = (c_{B2})^{D_A} \bmod N_A$

$(key_B, IV_B) = (m_{A2})^{E_B} \bmod N_B$

$m_{B2} = (c_{A2})^{D_B} \bmod N_B$

$(key_A, IV_A) = (m_{B2})^{E_A} \bmod N_A$

c_{Ai} →

← c_{Bi}

$c_{Ai} = d_{Ai} \otimes r_{Ai}$

$d_{Bi} = c_{Bi} \otimes r_{Bi}$

$c_{Bi} = d_{Bi} \otimes r_{Bi}$

$d_{Ai} = c_{Ai} \otimes r_{Ai}$

Probleme: Autosynchronisation

- Cipher Feed Back mode (CFB)