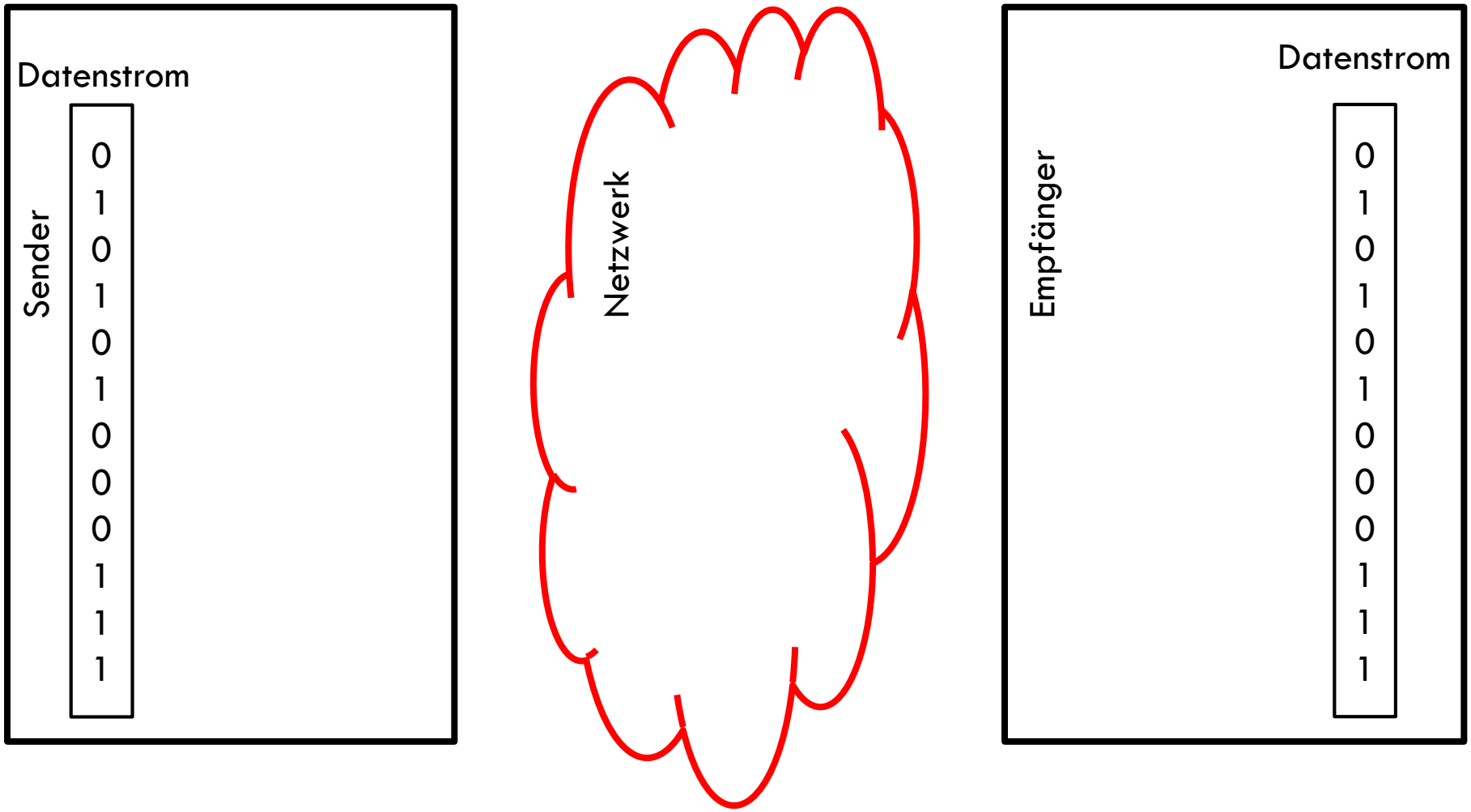
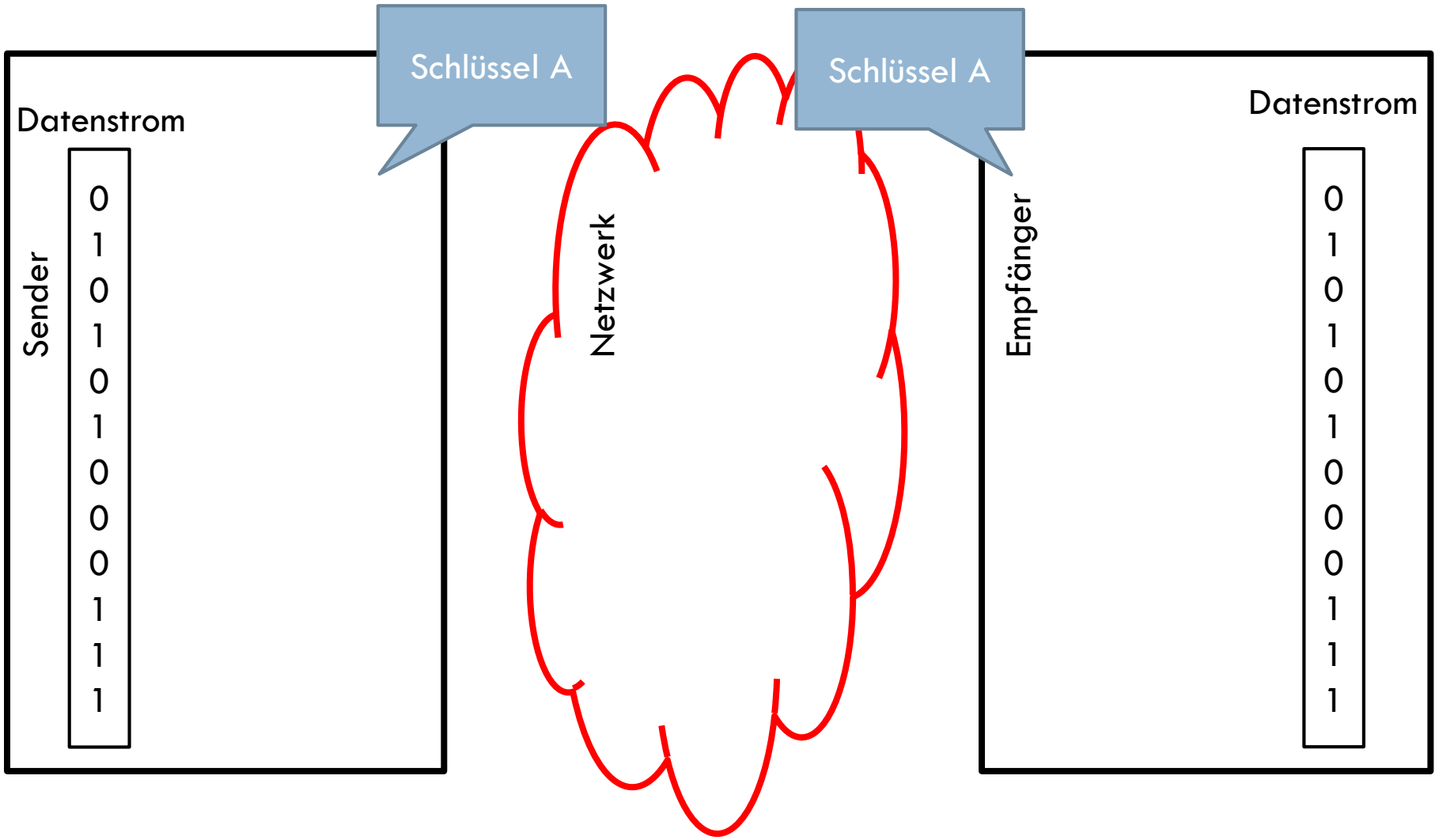
HACKERPRAKTIKUM
PART II: ANGRIFF AUF WEP

Lukas Jung, Marc Narres-Schulz, Oliver Sänger, Tobias Zeimetz

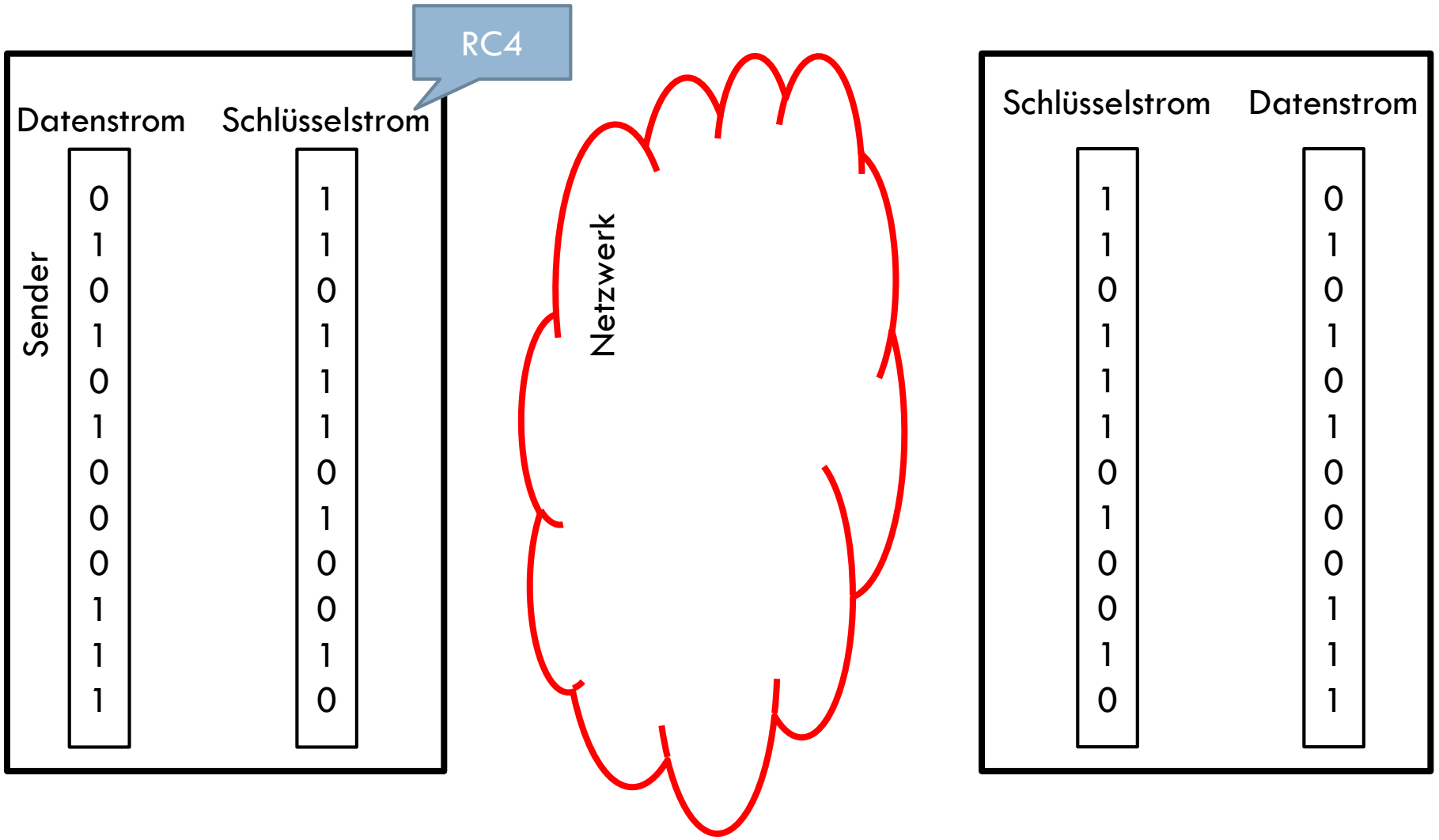
Funktionsweise



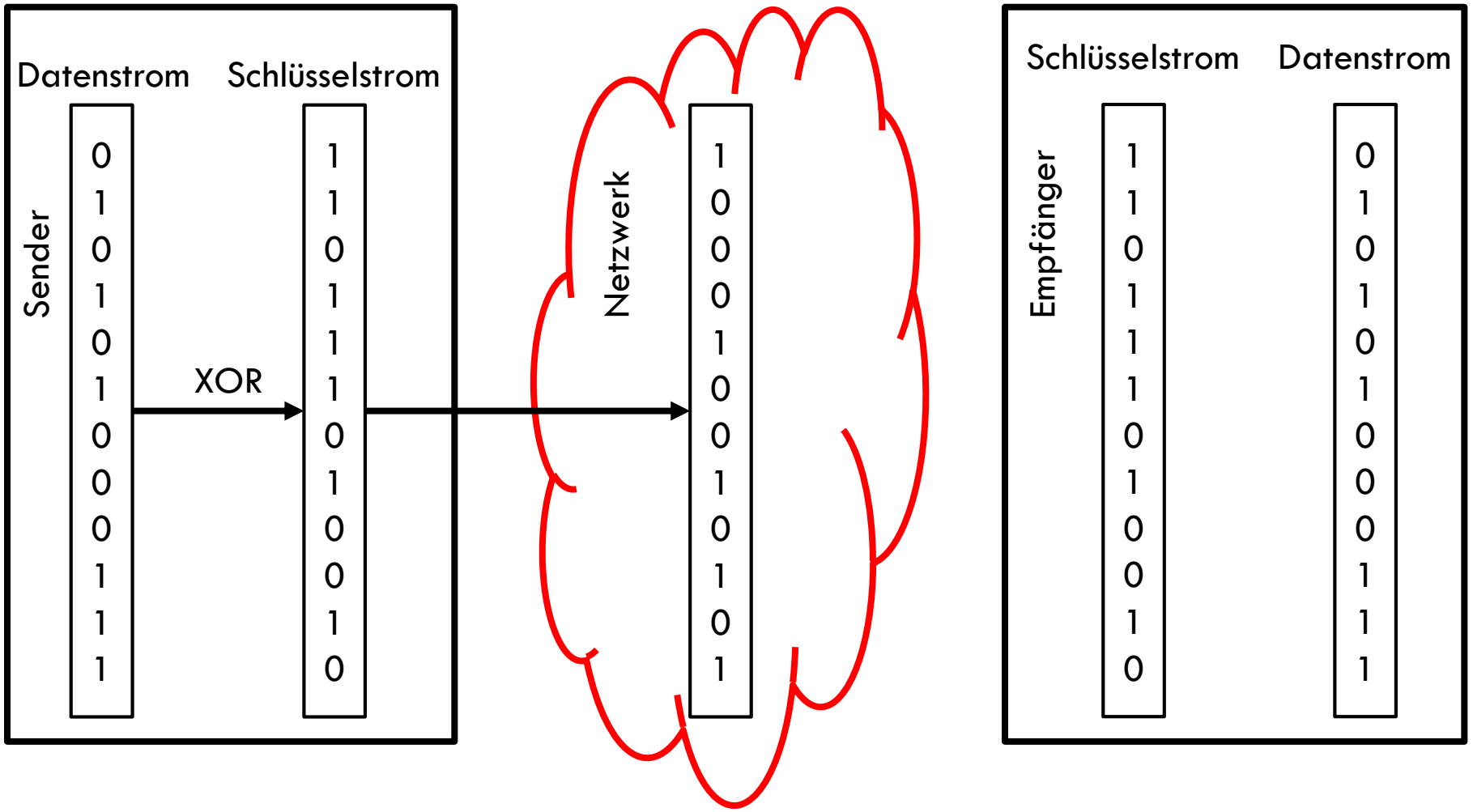
Funktionsweise



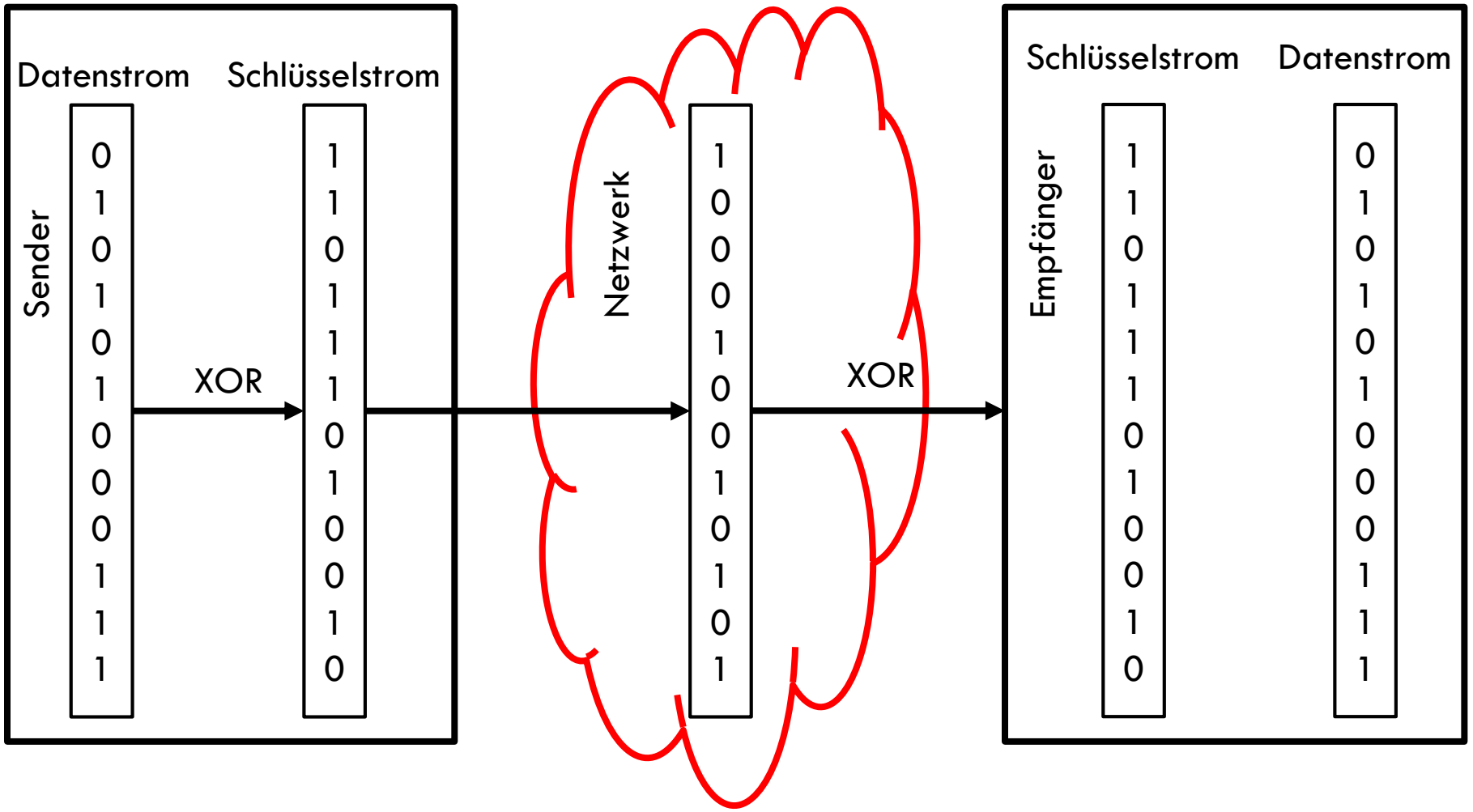
Funktionsweise



Funktionsweise



Funktionsweise



Aufbau eines WEP-Pakets

- IV: Initialisierungsvektor
- M : Nachricht
- c(M): Prüfsumme (stellt Intigrität sicher)
- RC4(IV | k): RC4-Chiffre der entsprechenden Länge

RC 4

Unter teilt sich in 2 Phasen

- Key Scheduling
- Pseudo Random Generation

RC 4 - Key Scheduling

Generierung einer S-Box

```
j = 0
S = [0,1,…,n-1]
for i from 0 to n-1 do
        j = (j+ S[i] + K[i mod l]) mod n
        swap(S[i], S[j])
end for
```

RC 4 - Key Scheduling

Generierung einer S-Box

```
j = 0
S = [0,1,…,n-1]
for i from 0 to n-1 do
        j = (j+ S[i] + K[i mod l]) mod n
        swap(S[i], S[j])
end for
```

Zufällig ?

RC 4 - Key Scheduling

Generierung einer S-Box

```
j = 0
S = [0,1,…,n-1]
for i from 0 to n-1 do
        j = (j+ S[i] + K[i mod l]) mod n
        swap(S[i], S[j])
end for
```

RC 4 - Key Scheduling

```
j = 0
S = [0,1,…,n-1]
for i from 0 to n-1 do
        j = (j+ S[i] + K[i mod l]) mod n
        swap(S[i], S[j])
end for
```

Schritt 1:

$j = 0 + S[0] + K[0]$

RC 4 - Key Scheduling

```
j = 0
S = [0,1,…,n-1]
for i from 0 to n-1 do
        j = (j+ S[i] + K[i mod l]) mod n
        swap(S[i], S[j])
end for
```

Schritt 1:

$j = 0 + S[0] + K[0]$

$j = K[0]$

$\rightarrow S[0] = S[K[0]]$

RC 4 - Key Scheduling

```
j = 0
S = [0,1,…,n-1]
for i from 0 to n-1 do
        j = (j+ S[i] + K[i mod l]) mod n
        swap(S[i], S[j])
end for
```

Schritt 2:

j = K[0]

RC 4 - Key Scheduling

```
j = 0
S = [0,1,…,n-1]
for i from 0 to n-1 do
        j = (j+ S[i] + K[i mod l]) mod n
        swap(S[i], S[j])
end for
```

Schritt 2:

j = K[0]

j = K[0] + S[1] + K[1]

Mit WK von 1 – 1/n

RC 4 - Key Scheduling

```
j = 0
S = [0,1,…,n-1]
for i from 0 to n-1 do
        j = (j+ S[i] + K[i mod l]) mod n
        swap(S[i], S[j])
end for
```

Schritt 2:

j = K[0]

j = K[0] + S[1] + K[1]

j = K[0] + 1 + K[1]

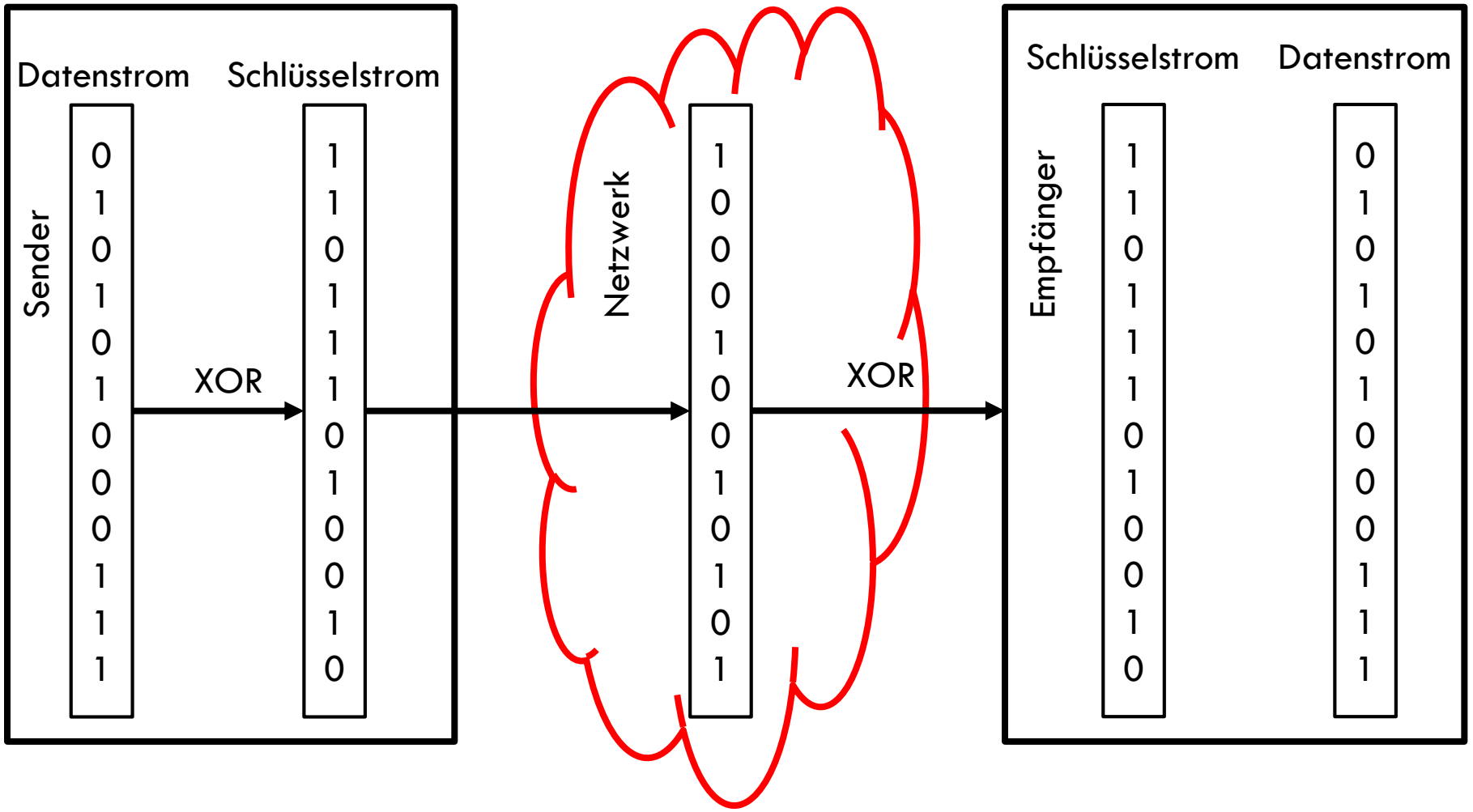
→ S[1] = S[K[0] + 1 + K[1]]

RC 4 - Pseudo Random Generation

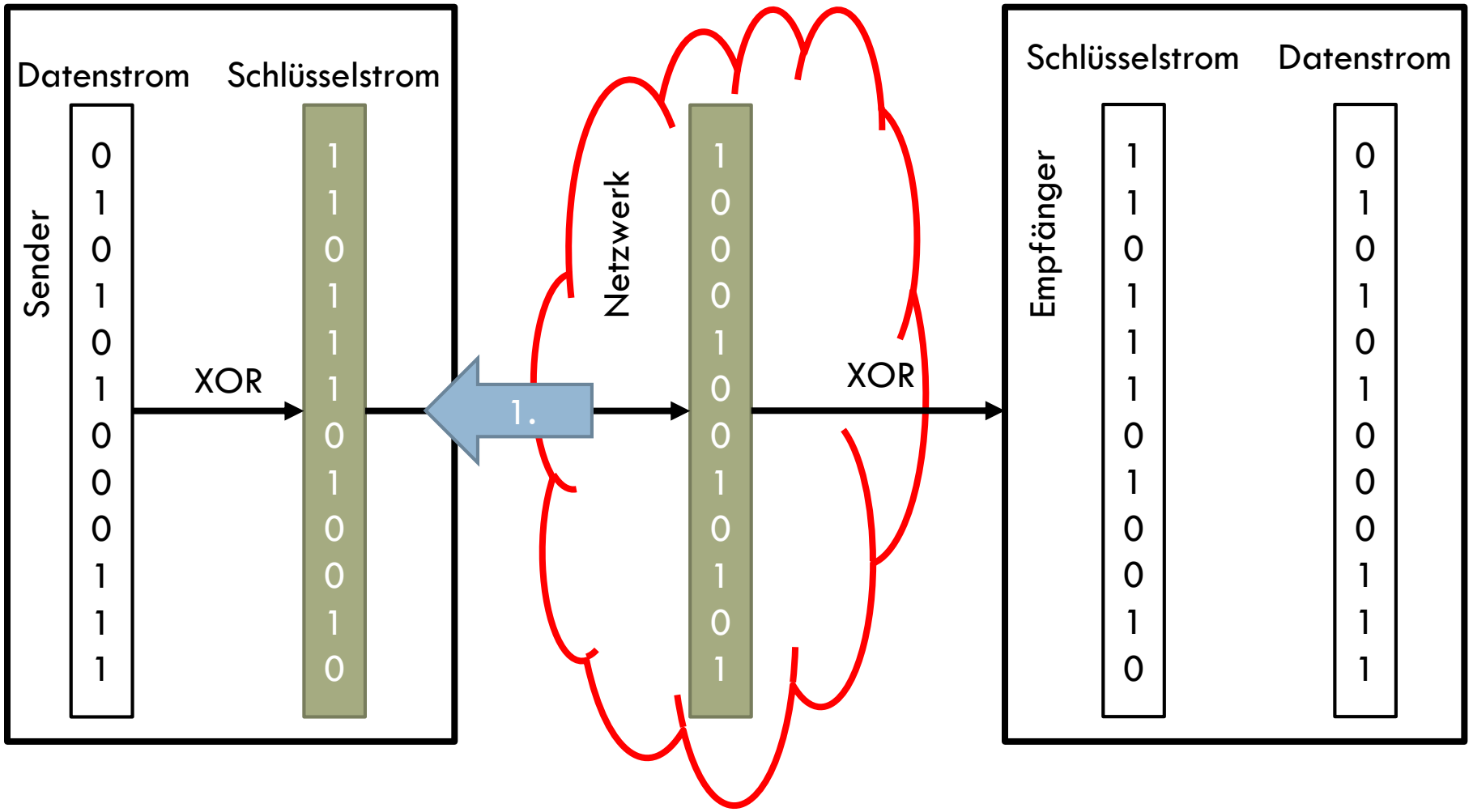
```
while 1:
       i = (i+1) % n
       j = (j +S[i]) % n
       swap(S[i],S[j])
       k = (S[i]+S[j]) % n
       yield S[k]
end loop
```

Nutzt die generierte S-Box zur Erzeugung des Schlüsselstroms

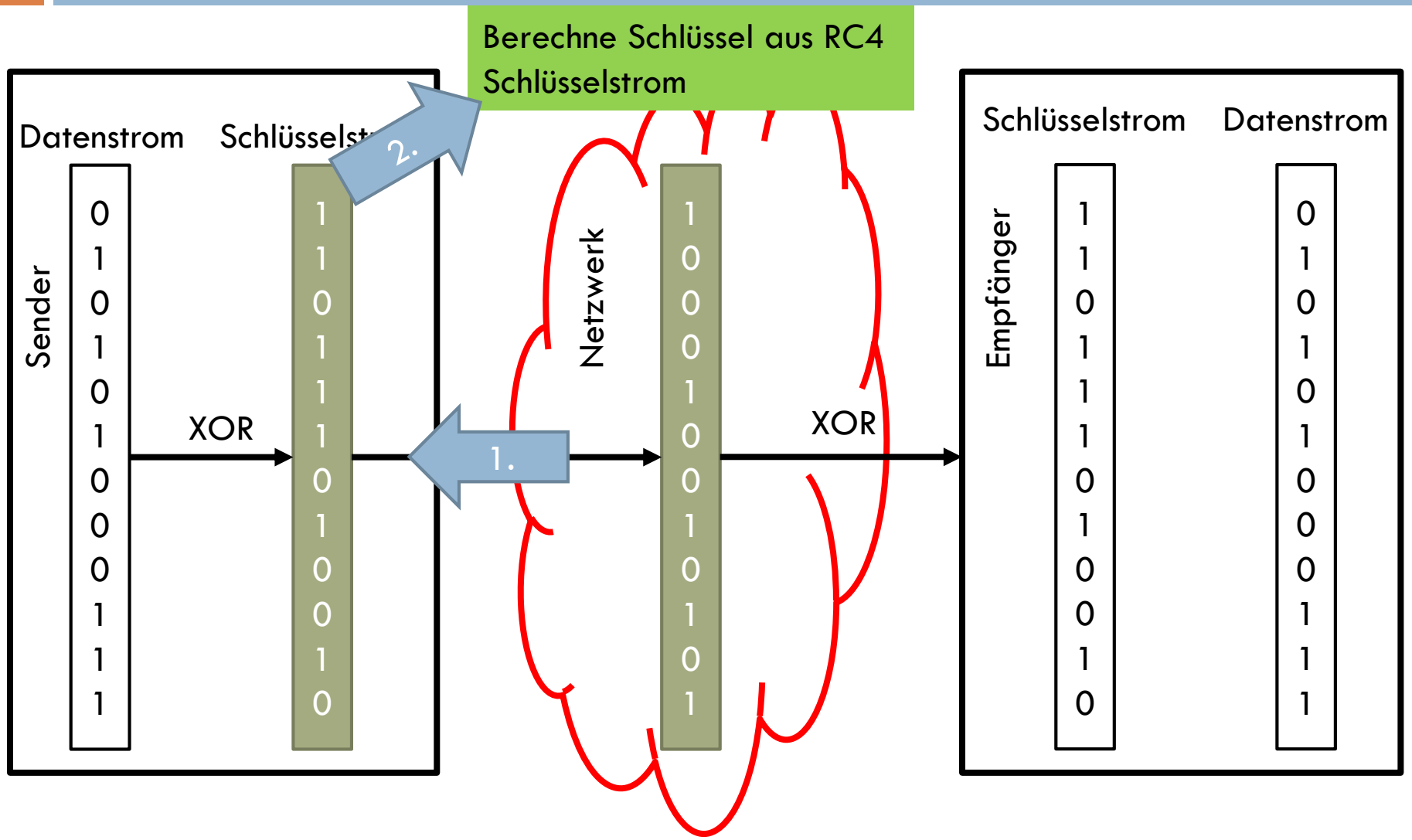
Der Angriff


Sender
Datenstrom
Schlüsselstrom
0 1 0 1 0 1 0 0 0 1 1 1
XOR
1 1 0 1 1 1 1 0 1 0 0 1 0
Netzwerk
1 0 0 0 1 0 0 1 0 1 0 1
XOR
Empfänger
Schlüsselstrom
Datenstrom
1 1 0 1 1 1 1 0 1 0 0 1 0
0 1 0 1 0 1 0 0 0 1 1 1

Der Angriff


Sender
Datenstrom
Schlüsselstrom
XOR
1.
Netzwerk
XOR
Empfänger
Schlüsselstrom
Datenstrom

Der Angriff


Berechne Schlüssel aus RC4 Schlüsselstrom
Datenstrom
Schlüsselstrom
Sender
0 1 0 1 0 1 0 0 0 1 1 1
XOR
1 1 0 1 1 1 1 0 1 0 0 1 0
2.
1.
Netzwerk
1 0 0 0 1 0 0 1 0 1 0 1
XOR
Schlüsselstrom
Datenstrom
Empfänger
1 1 0 1 1 1 1 0 1 0 0 1 0
0 1 0 1 0 1 0 0 0 1 1 1

Der Angriff

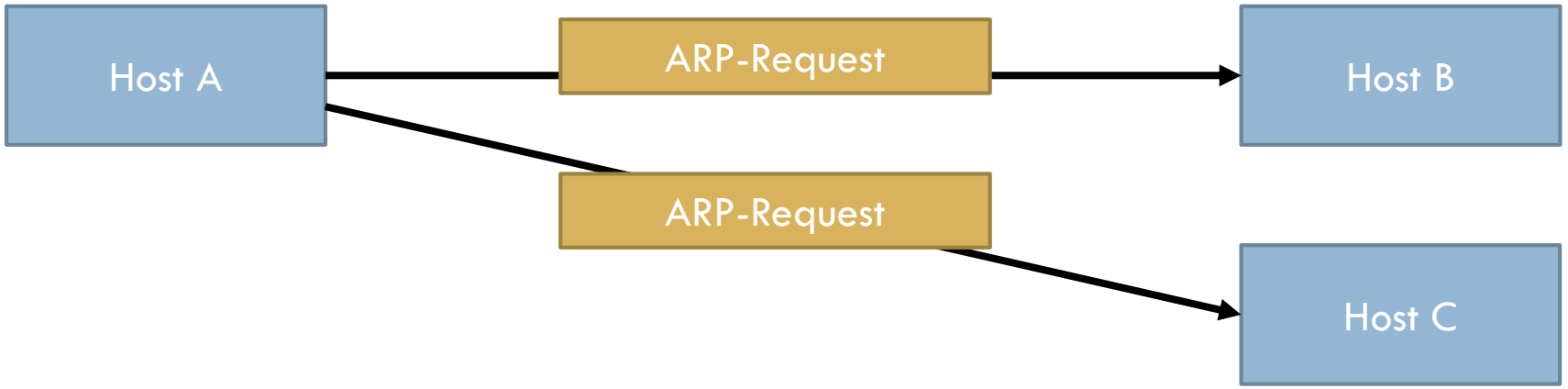
1. Von Chiffrestrom auf Schlüsselstrom schließen
    1. Hierzu: Sammle Klartext und Chiffretext Paare mittels ARP-Reinjection
2. Nutze Schwächen in RC4 um den Schlüssel zu extrahieren

Von Chiffrestrom zu Schlüsselstrom

- Nutze Eigenschaften des Address Resolution Protokolls
- Dient zur Addressauflösung im Netzwerk
- Pakete von fester Länge

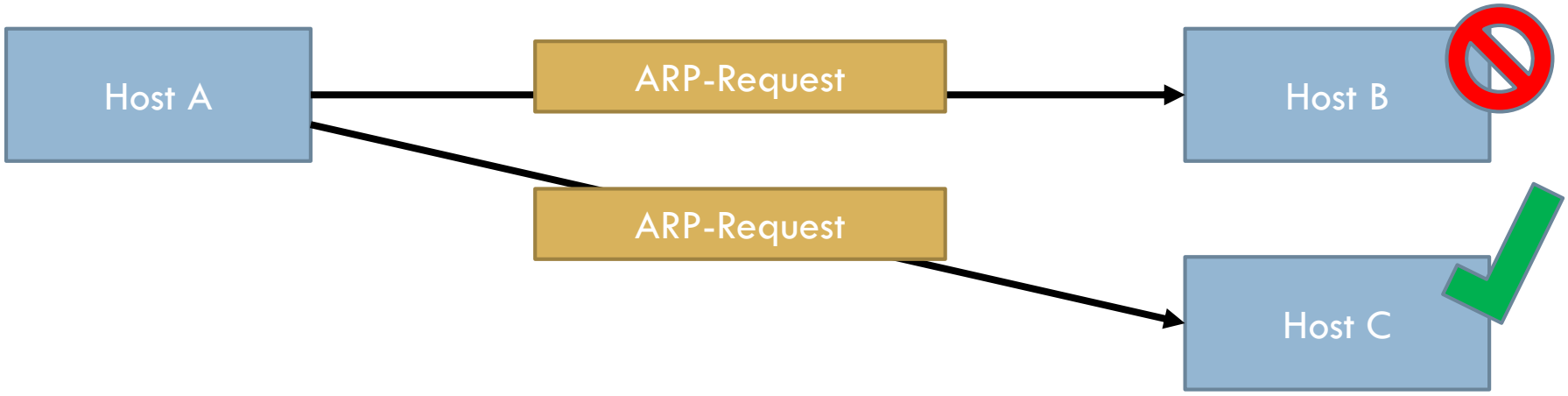
Von Chiffrestrom zu Schlüsselstrom

Funktionsweise ARP



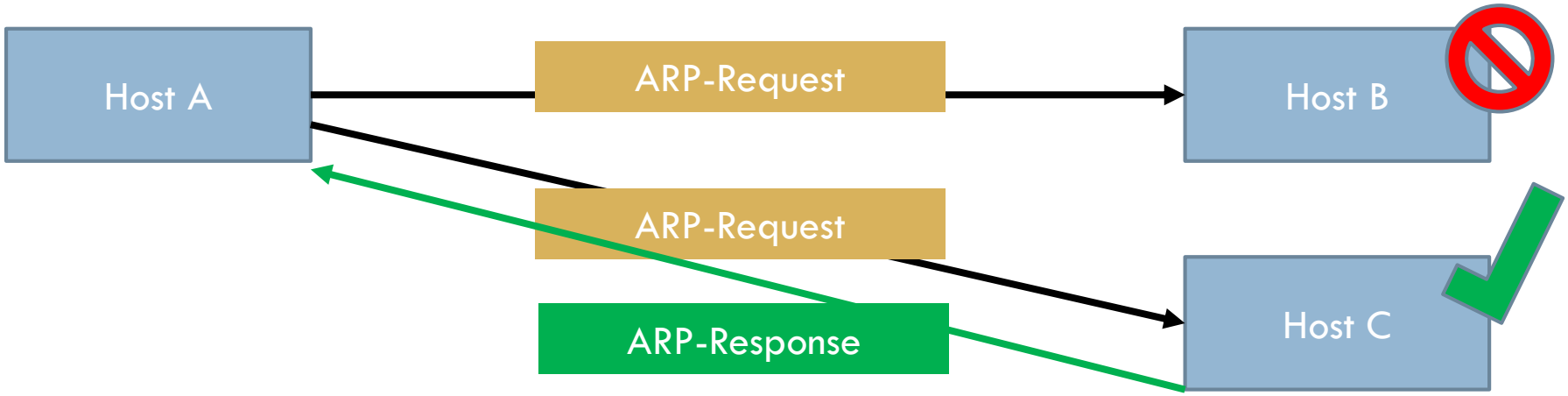
Von Chiffrestrom zu Schlüsselstrom

Funktionsweise ARP



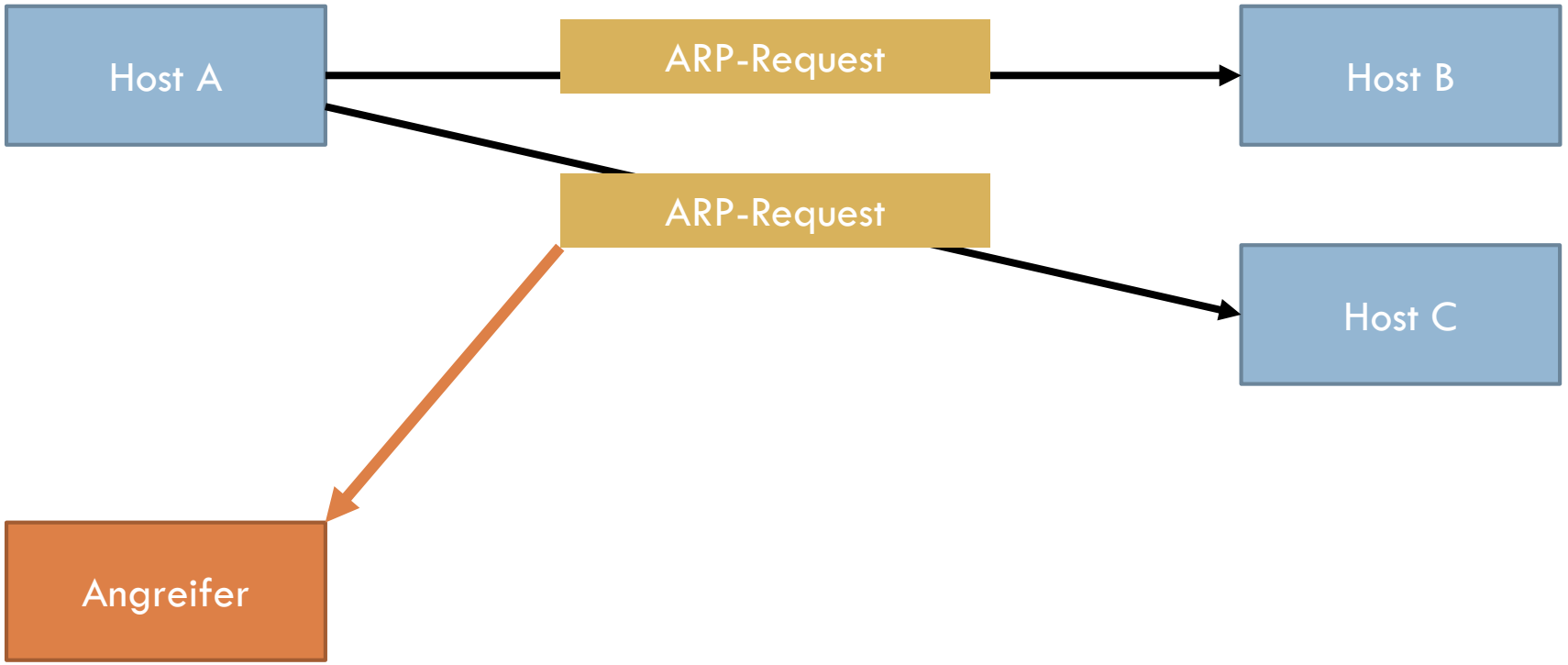
Von Chiffrestrom zu Schlüsselstrom

Funktionsweise ARP



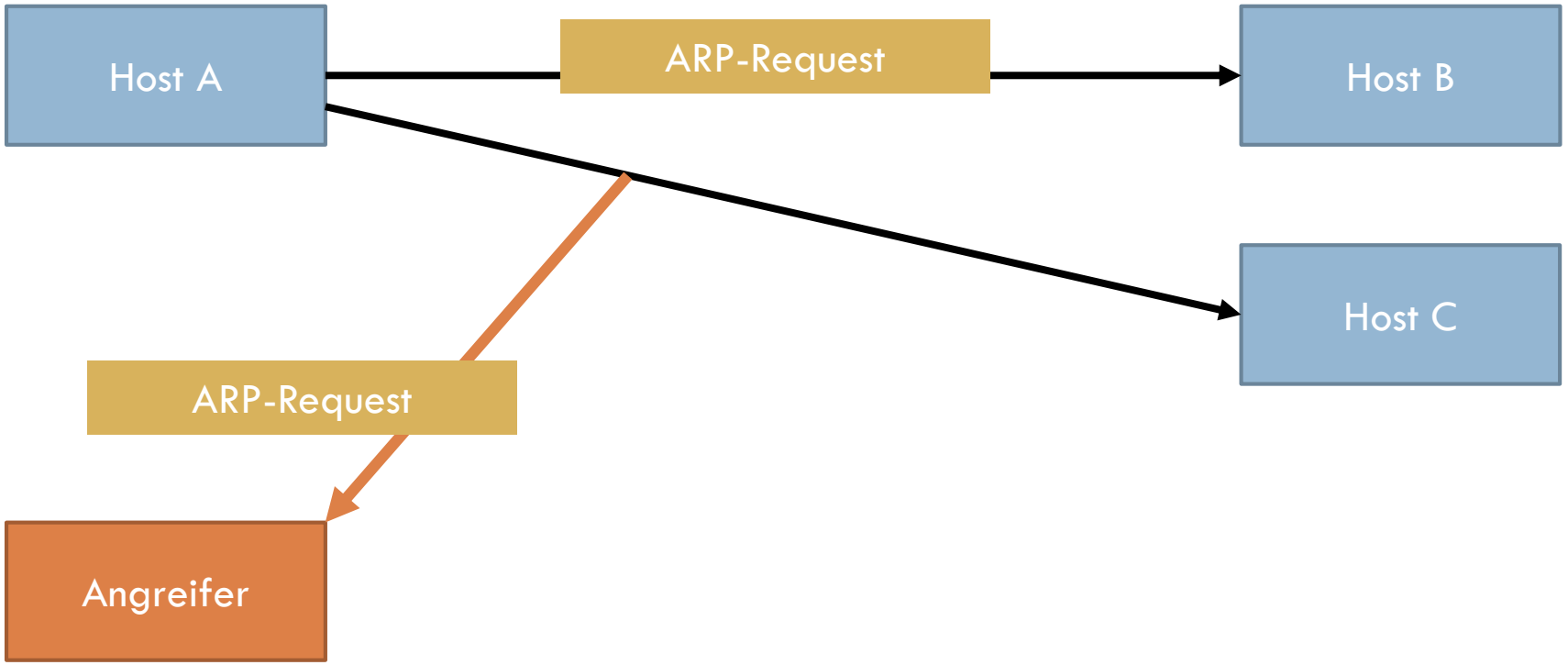
Von Chiffrestrom zu Schlüsselstrom

ARP-Reinjection



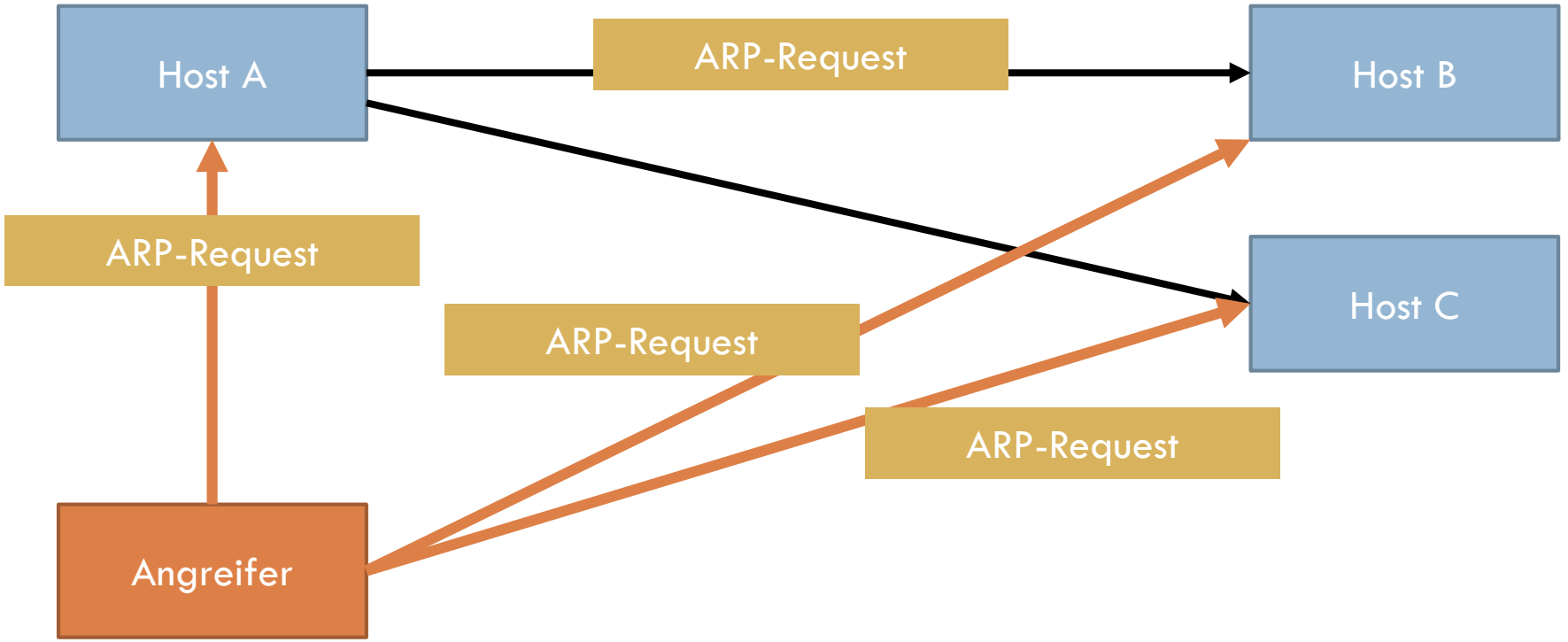
Von Chiffrestrom zu Schlüsselstrom

ARP-Reinjection



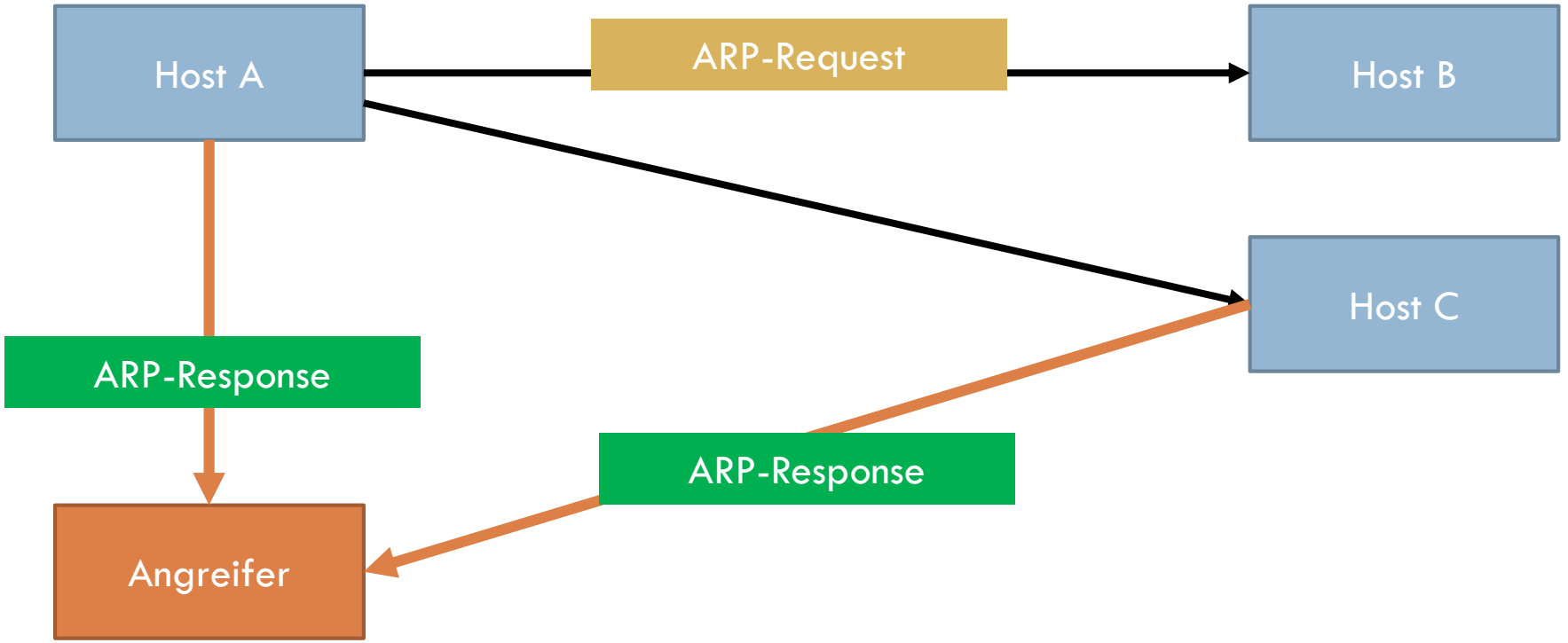
Von Chiffrestrom zu Schlüsselstrom

ARP-Reinjection



Von Chiffrestrom zu Schlüsselstrom

ARP-Reinjection



Von Chiffrestrom zu Schlüsselstrom

ARP-Reinjection

Nun haben wir 100.000 Pakete gesammelt.

Von Chiffrestrom zu Schlüsselstrom

ARP-Reinjection

Nun haben wir 100.000 Pakete gesammelt.

→ Und jetzt?

Von Chiffrestrom zu Schlüsselstrom

ARP-Reinjection

ARP-Pakete beginnen IMMER mit den 18 gleichen Bytes:

AA AA 03 00 00 00 08 06 00 01 08 00 06 04 01

Von Chiffrestrom zu Schlüsselstrom

ARP-Reinjection
ARP-Pakete beginnen IMMER mit den 18 gleichen Bytes:

AA AA 03 00 00 00 08 06 00 01 08 00 06 04 01

Somit kennen wir nun 100.000 (Klartext,Chiffretext)-Paare

→ *Durch einfaches XOR lässt sich der Schlüsselstrom regenerieren*

Der Angriff

1. ~~Von Chiffrestrom auf Schlüsselstrom schließen~~
   1. ~~Hierzu: Sammle Klartext und Chiffretext Paare mittels ARP-Reinjection~~
2. Nutze Schwächen in RC4 um den Schlüssel zu extrahieren

Auf Hauptschlüssel schließen

Dem WEP Paket steht ein 3 Byte IV vor

Somit kann der Zustand der S-Box bis zum 3. Schritt simuliert werden

$$K[i] = S_{i-1}^{-1}[i - [X[i-1]] - (j_{i-1} + S_{i-1}[i]) mod\ n$$

Auf Hauptschlüssel schließen

Dem WEP Paket steht ein 3 Byte IV vor

Somit kann der Zustand der S-Box bis zum 3. Schritt simuliert werden

$$K[i] = S_{i-1}^{-1}[i - [X[i-1]] - (j_{i-1} + S_{i-1}[i]) mod\ n$$
$$K[3] = S_2^{-1}[3 - [X[2]] - (j_2 + S_2[3]) mod\ n$$

➔ Abhängig von den vorherigen S-Box-Zuständen

Auf Hauptschlüssel schließen

Mache diese Berechnung für alle gesammelten Pakete und gibt die “most common”-Bytes aus

Auf Hauptschlüssel schließen

Mache diese Berechnung für alle gesammelten Pakete und gibt die “most common”-Bytes aus

Test:

Verschlüssle eins der gesammelten Pakete

Auf Hauptschlüssel schließen

Mache diese Berechnung für alle gesammelten Pakete und gibt die “most common”-Bytes aus

Test:

Verschlüssle eins der gesammelten Pakete

→ Erfolg? Glückwunsch

Auf Hauptschlüssel schließen

Mache diese Berechnung für alle gesammelten Pakete und gibt die “most common”-Bytes aus

Test:

Verschlüssle eins der gesammelten Pakete

→ Erfolg? Glückwunsch

→ Kein erfolg? Starte mit mehr Tupeln

Nachteile unseres Angriffes

- “langsam”: circa 5 Minuten für WEP128
- Iteratives Verfahren
- Nicht sehr fehlertolerant

Verbesserung unseres Angriffes

- Berechnung von Votes

$$\sigma_i \approx S_3^{-1}\left[(3 + \mathrm{i}) - \mathrm{X}[2 + \mathrm{i}]\right] - \left(j_3 + \sum_{l=3}^{i+3} S_3[l]\right)$$

- S_3^{-1}: Invertierte S-Box an Stelle 3

Verbesserung unseres Angriffes

- Berechnung von Votes

$$\sigma_i \approx S_3^{-1}\left[(3 + \mathrm{i}) - \mathrm{X}[2 + \mathrm{i}]\right] - \left(j_3 + \sum_{l=3}^{i+3} S_3[l]\right)$$

- S_3^{-1}: Invertierte S-Box an Stelle 3
- X: Stromchiffre

Verbesserung unseres Angriffes

- Berechnung von Votes

$$\sigma_i \approx S_3^{-1}\left[(3+\mathrm{i}) - \mathrm{X}[2+\mathrm{i}]\right] - \left(j_3 + \sum_{l=3}^{i+3} S_3[l]\right)$$

- S_3^{-1}: Invertierte S-Box an Stelle 3
- X: Stromchiffre
- S_3: S-Box an Stelle 3

Verbesserung unseres Angriffes

- Berechnung von Votes

$$\sigma_i \approx S_3^{-1}\left[(3+\mathrm{i}) - \mathrm{X}[2+\mathrm{i}]\right] - \left(j_3 + \sum_{l=3}^{i+3} S_3[l]\right)$$

- S_3^{-1}: Invertierte S-Box an Stelle 3
- X: Stromchiffre
- S_3: S-Box an Stelle 3
- j_3: Index aus der S-Box an Stelle 3

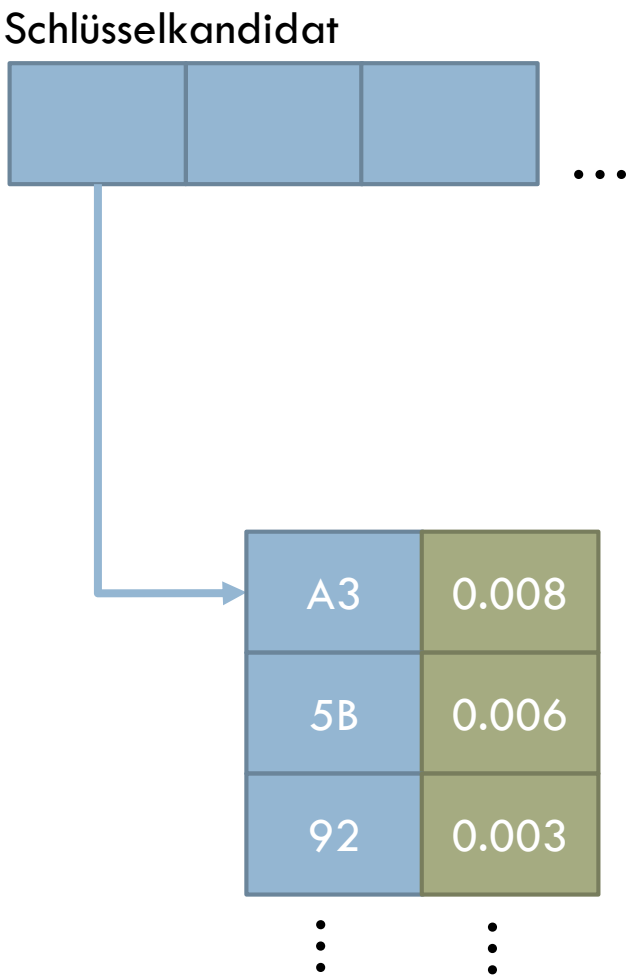
Verbesserung unseres Angriffes

- Berechnung von Votes

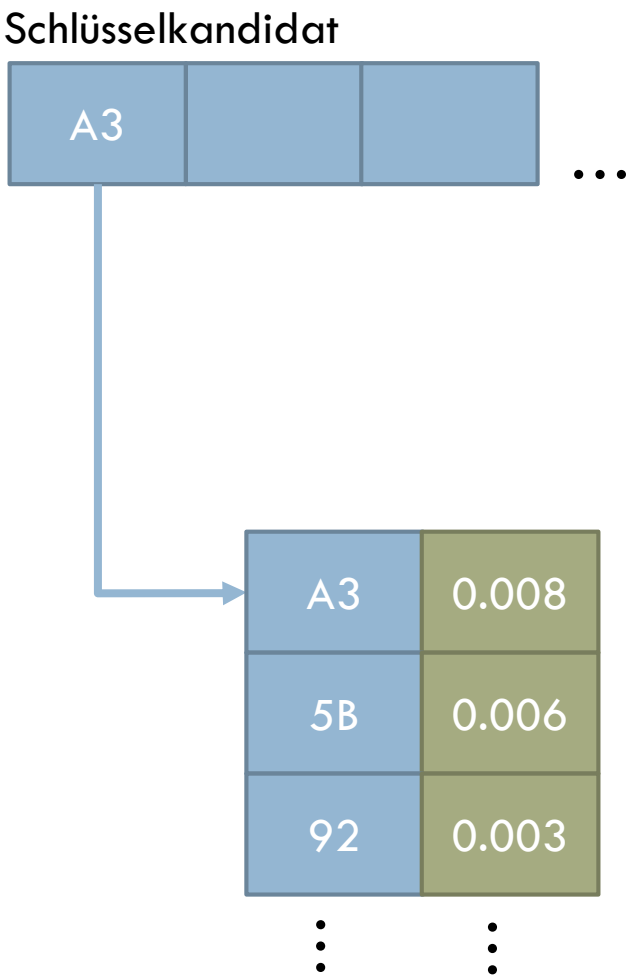
$$\sigma_i \approx S_3^{-1}\left[(3+\mathrm{i}) - \mathrm{X}[2+\mathrm{i}]\right] - \left(j_3 + \sum_{l=3}^{i+3} S_3[l]\right)$$

- Votes hängen von keinen vorherigen Votes ab und kann parallel berechnet werden
- Votes werden an Ihrer Position in Mengen gespeichert

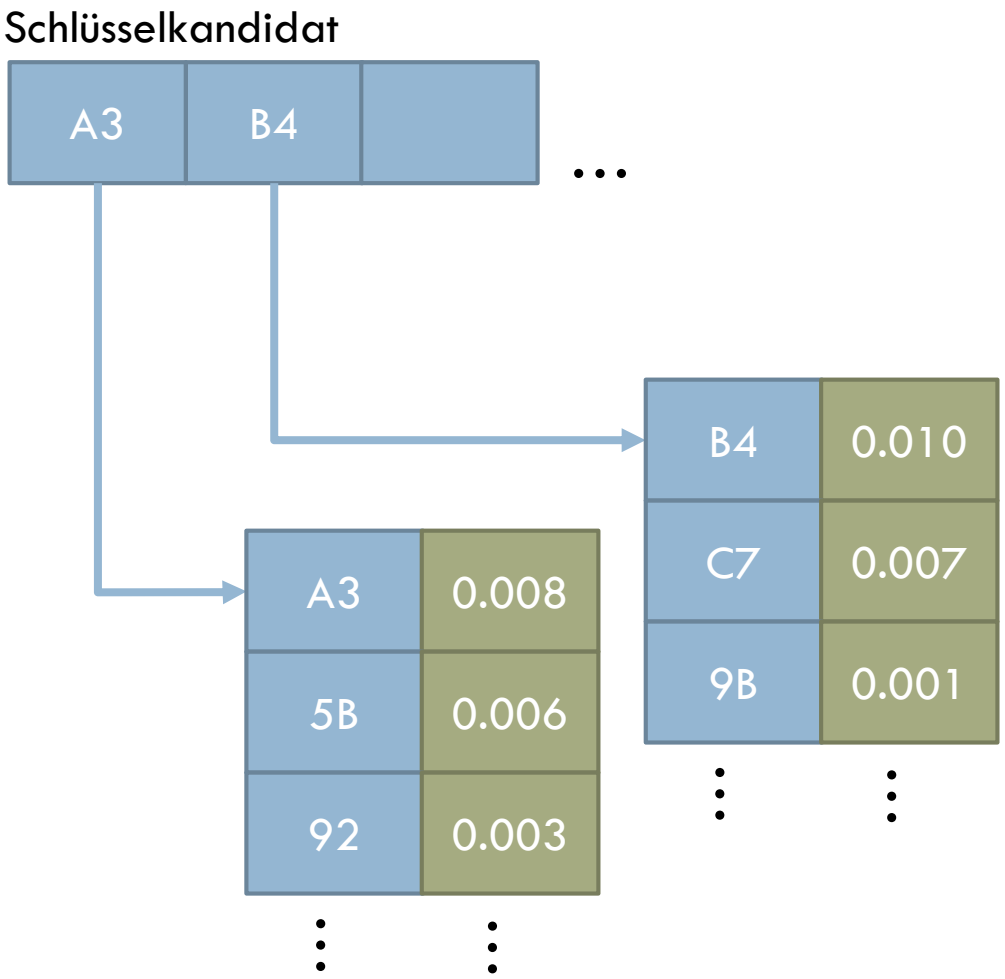
Verbesserung unseres Angriffes



Verbesserung unseres Angriffes



Verbesserung unseres Angriffes



Verbesserung unseres Angriffes

Schlüsselkandidat

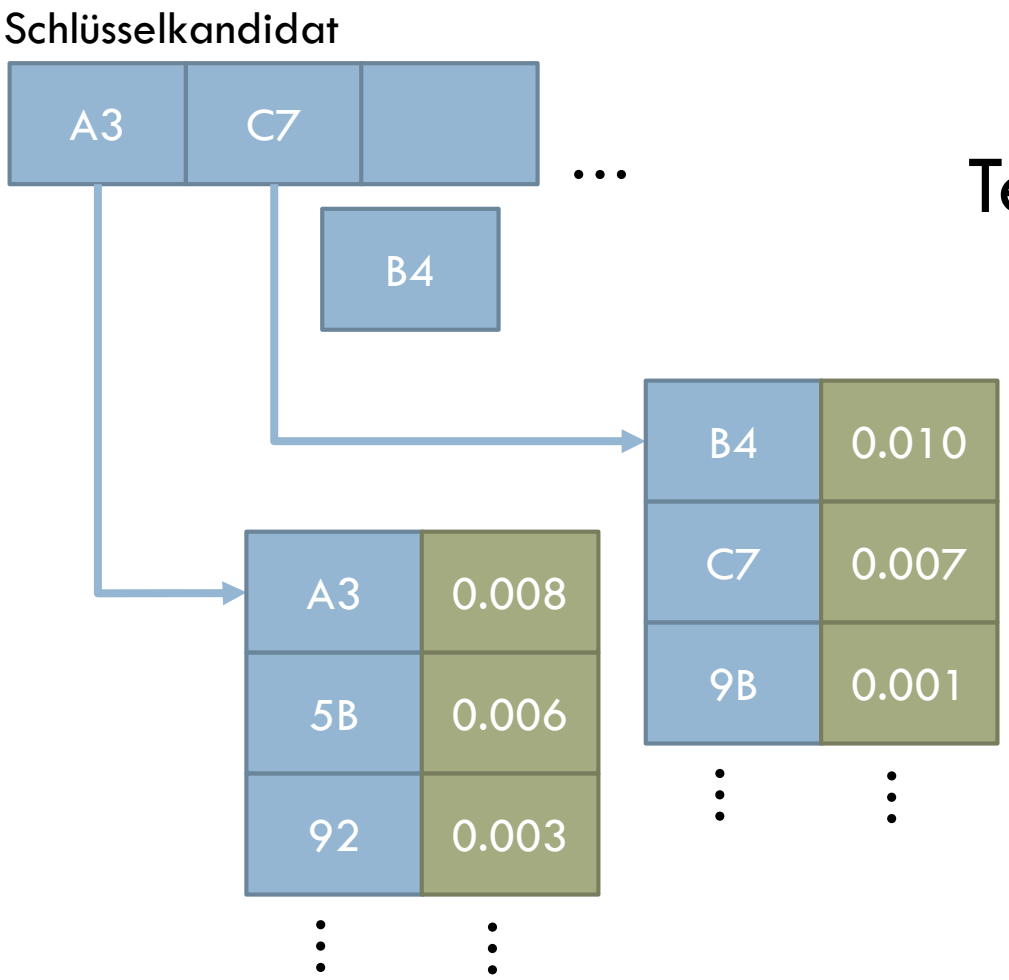
Teste den Schlüssel

→ Erfolgreich? – Glückwunsch!

→ Nicht Erfolgreich?

→ Teste nächstwahrscheinlichen Schlüssel

Verbesserung unseres Angriffes



Teste erneut!

Nachteile unseres Angriffes

- “langsam”: circa 5 Minuten für WEP128
- Iteratives Verfahren
- Nicht sehr fehlertolerant

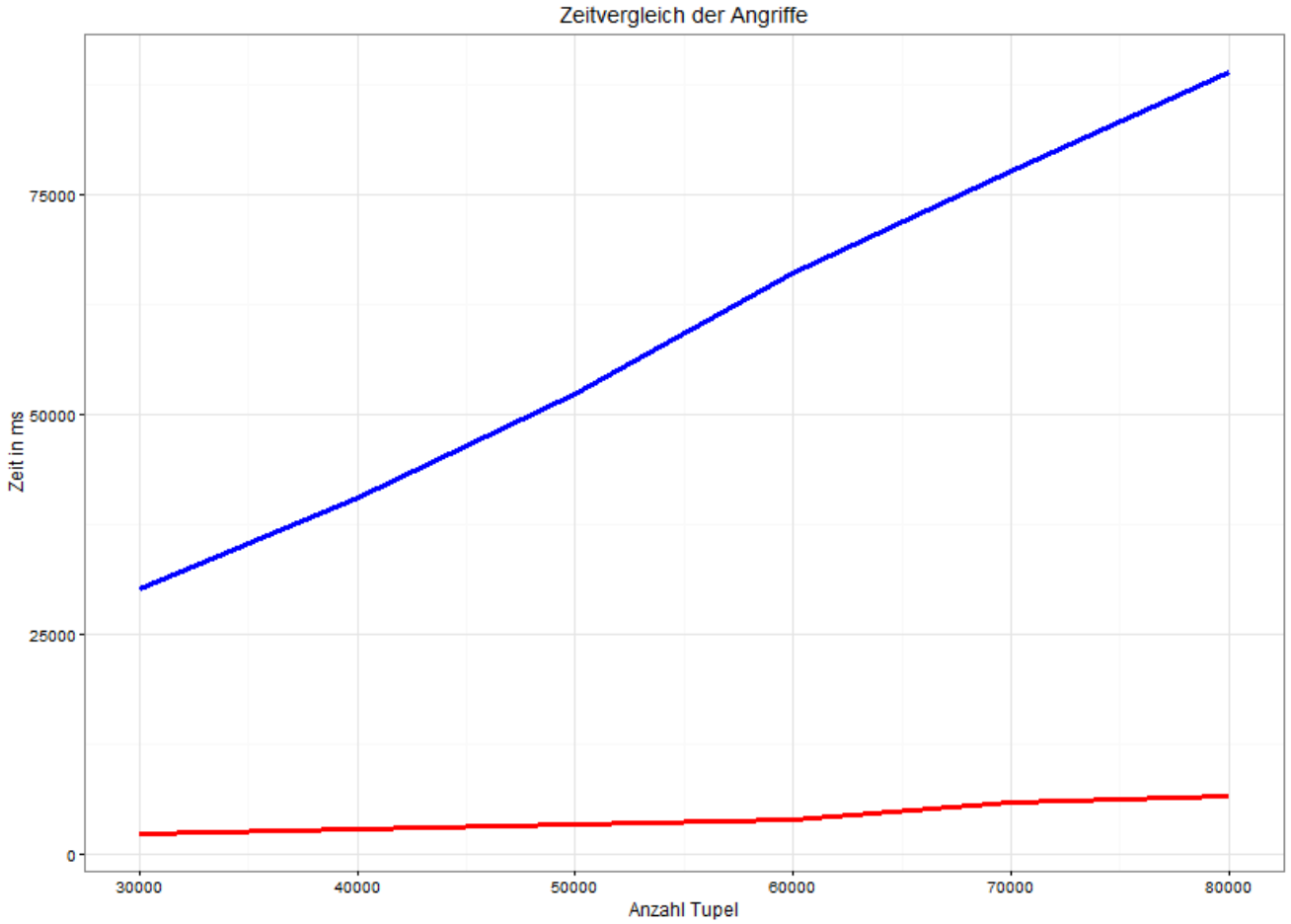
Nachteile unseres Angriffes

- “langsam”: circa 5 Minuten für WEP128
- ~~Iteratives Verfahren~~
- ~~Nicht sehr fehlertolerant~~

Performancevorteile

	Angriff Klein auf WEP 40	Verbesserter Angriff auf WEP 104
Dauer	~ 90 Sekunden	~ 6.5 Sekunden
Erfolgswahrscheinlichkeit	74 %	76 %
Untersuchte Tupel	7000	4000

Performancevorteile



Nachteile unseres Angriffes

- ~~“langsam”: circa 5 Minuten für WEP128~~
- ~~Iteratives Verfahren~~
- ~~Nicht sehr fehlertolerant~~