

# Block Chiffre und der *Data Encryption Standard (DES)*

**Dozent:** Prof. Dr. Michael Eichberg

**Version:** 1.0

**Basierend auf:** *Cryptography and Network Security - Principles and Practice, 8th Edition, William Stallings*



1

---

**Folien:** **HTML:** <https://delors.github.io/sec-blockchiffre/folien.de.rst.html>  
**PDF:** <https://delors.github.io/sec-blockchiffre/folien.de.rst.html.pdf>  
**Fehler melden:** <https://github.com/Delors/delors.github.io/issues>

# Stromchiffre

- Verschlüsselt einen digitalen Datenstrom ein Bit oder ein Byte nach dem anderen.  
Beispiele: Autokeyed Vigenère-Chiffre und Vernam-Chiffre.
- Im Idealfall wird (würde) ein One-Time-Pad verwendet werden. Der Schlüsselstrom wäre genauso lang wie der Bitstrom des Klartextes.

- Wenn der kryptografische Schlüsselstrom zufällig ist, kann diese Chiffre auf keine andere Weise als durch die Beschaffung des Schlüsselstroms geknackt werden.
- Der Schlüsselstrom muss beiden Nutzern im Voraus über einen unabhängigen und

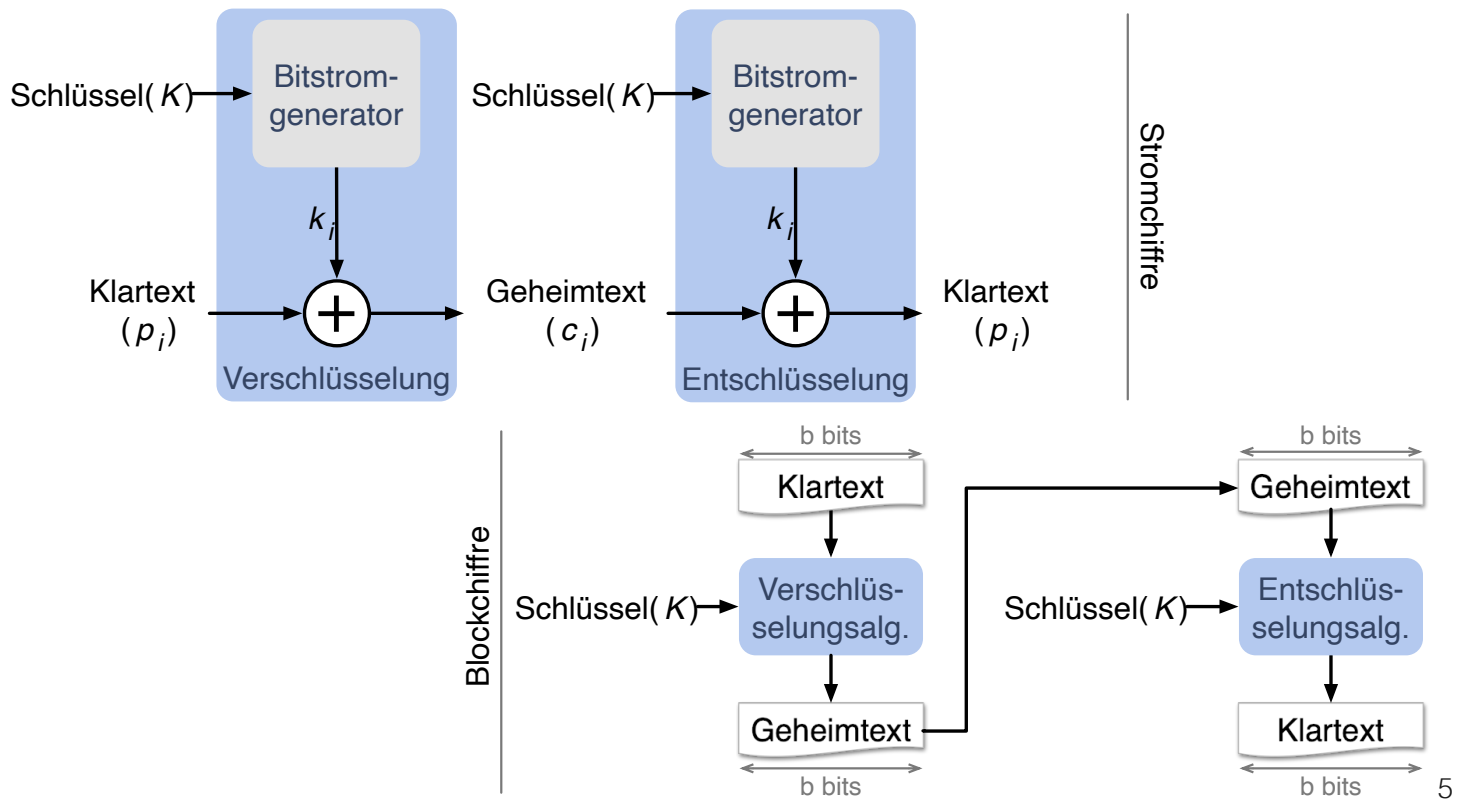
# Stromchiffre

- Aus praktischen Gründen muss der Bitstromgenerator als algorithmisches Verfahren implementiert werden, damit der kryptografische Bitstrom von beiden Benutzern erzeugt werden kann.
  - Es muss rechnerisch praktisch unmöglich sein, zukünftige Teile des Bitstroms auf der Grundlage früherer Teile des Bitstroms vorherzusagen.
  - Die beiden Benutzer müssen nur den erzeugenden Schlüssel sicher miteinander teilen, damit jeder den Schlüsselstrom erzeugen kann.

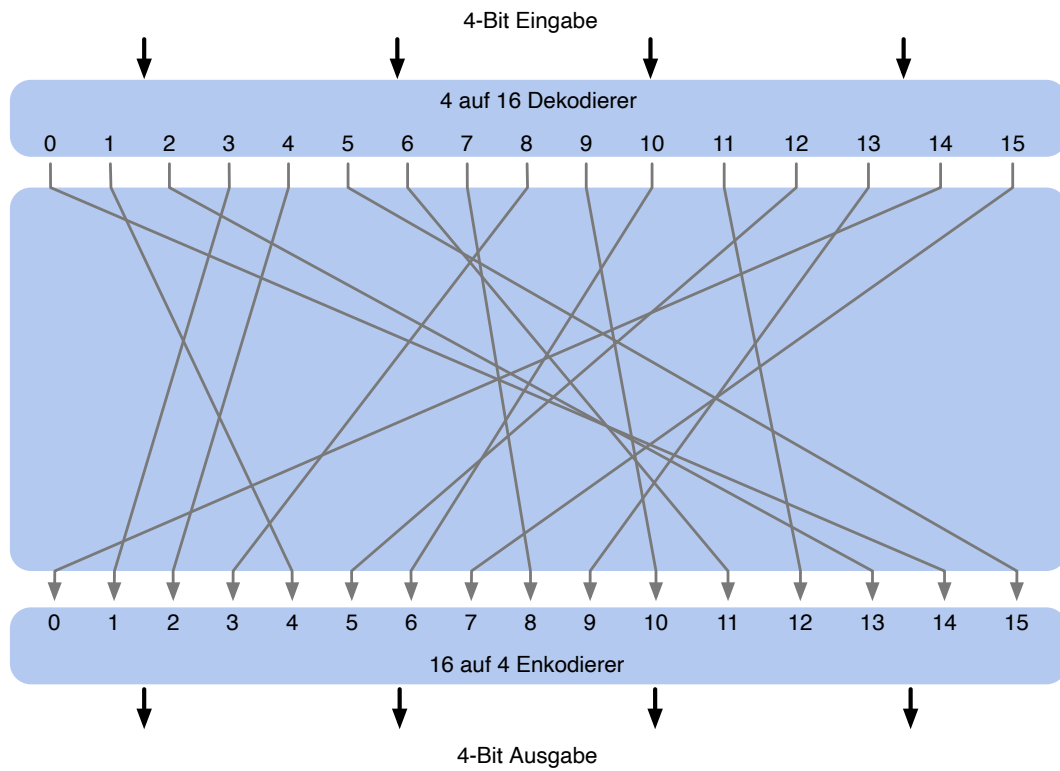
# Blockchiffre

- Ein Klartextblock wird als Ganzes behandelt und verwendet, um einen gleich langen Chiffretextblock zu erzeugen.
- In der Regel wird eine Blockgröße von 64 oder 128 Bit verwendet.
- Wie bei einer Stromchiffre teilen sich die beiden Benutzer einen symmetrischen Chiffrierschlüssel.
- Die meisten netzbasierten Anwendungen, die auf symmetrische Verschlüsselung setzen, verwenden Blockchiffren.

# Stromchiffre vs. Blockchiffre



# Allgemeine n-Bit-n-Bit-Blocksubstitution (n = 4)



# Verschlüsselungs- und Entschlüsselungstabelle für eine Substitutions-Chiffre

## Verschlüsselungstabelle

|            |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
|------------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| Klartext   | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
| Geheimtext | 1110 | 0100 | 1101 | 0001 | 0010 | 1111 | 1011 | 1000 | 0011 | 1010 | 0110 | 1100 | 0101 | 1001 | 0000 | 0111 |

## Entschlüsselungstabelle

|            |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
|------------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| Geheimtext | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
| Klartext   | 1110 | 0011 | 0100 | 1000 | 0001 | 1100 | 1010 | 1111 | 0111 | 1101 | 1001 | 0110 | 1011 | 0010 | 0000 | 0101 |

# Feistel-Chiffre

Feistel schlug die Verwendung einer Chiffre vor, bei der sich Substitutionen und Permutationen abwechseln.

## Substitutionen

Jedes Klartextelement oder jede Gruppe von Elementen wird eindeutig durch ein entsprechendes Chiffretextelement oder eine entsprechende Gruppe von Elementen ersetzt.

## Permutation

Bei einer Permutation werden keine Elemente hinzugefügt, gelöscht oder ersetzt, sondern die Reihenfolge, in der die Elemente in einer Folge erscheinen, wird geändert.



# Feistel-Chiffre - Hintergrund

- Hierbei handelt es sich um eine praktische Anwendung eines Vorschlags von Claude Shannon zur Entwicklung einer Chiffre, bei der sich *Konfusions- und Diffusionsfunktionen* abwechseln.
- Dieser Aufbau wird von vielen bedeutenden (Twofish, Blowfish, Serpent, Mars) - im Einsatz befindlichen - symmetrischen Blockchiffren verwendet.

## Diffusion und Konfusion

- Begriffe, die von Claude Shannon eingeführt wurden, um die beiden grundlegenden Bausteine für jedes kryptografische System zu erfassen.
- Shannons Anliegen war es, die auf statistischer Analyse beruhende Kryptoanalyse zu vereiteln.

Blowfish ist zum Beispiel die Basis für das Hashingverfahren **bcrypt**, welches für Passworthashing verwendet wird.

# Diffusion

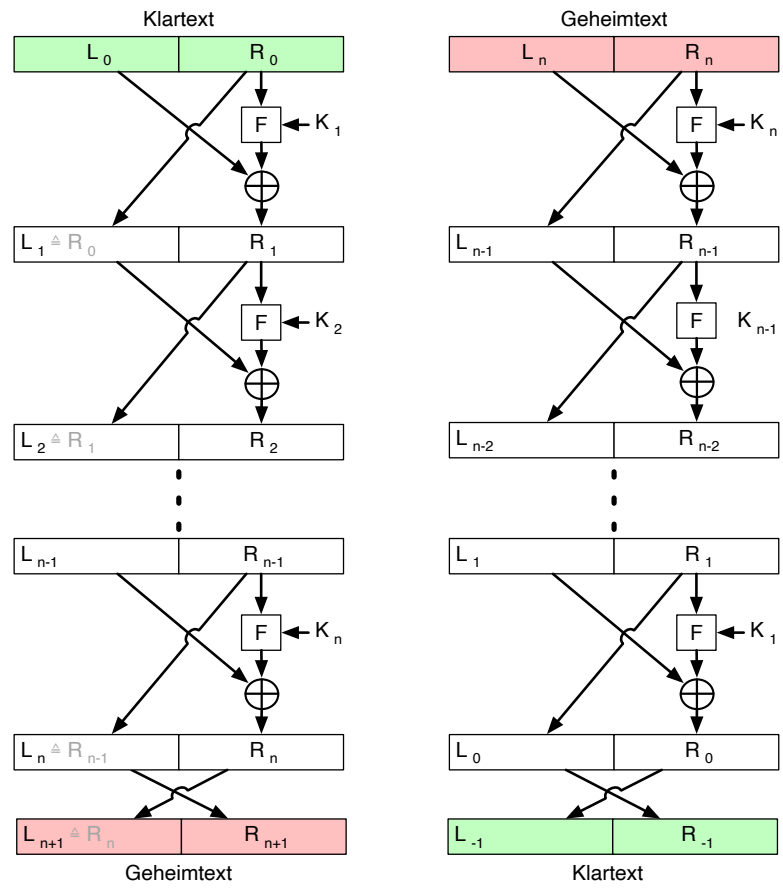
- Die statistische Struktur des Klartextes wird in weitreichende Statistiken des Chiffretextes überführt, d. h. die statistische Beziehung zwischen Klartext und Chiffretext wird so komplex wie möglich.
- Dies wird dadurch erreicht, dass jede Klartextziffer *den Wert vieler Chiffretextziffern* beeinflusst.  
(„Lawineneffekt“)
- Die Diffusion kann z.B. durch *Permutationen* erreicht werden.

# Konfusion

- Versucht, die Beziehung zwischen den Statistiken des Chiffriertextes und dem Wert des Chiffrierschlüssels so komplex wie möglich zu gestalten, d.h. eine einzige Änderung des Chiffrierschlüssels sollte viele Bits des Chiffriertextes beeinflussen.
- Selbst wenn der Angreifer die Statistik des Chiffretextes einigermaßen in den Griff bekommt, ist die Art und Weise, wie der Schlüssel verwendet wurde, um diesen Chiffretext zu erzeugen, so komplex, dass es schwierig ist, den Schlüssel abzuleiten.
- Die Verwirrung kann z.B. durch *Substitutionen* realisiert werden.

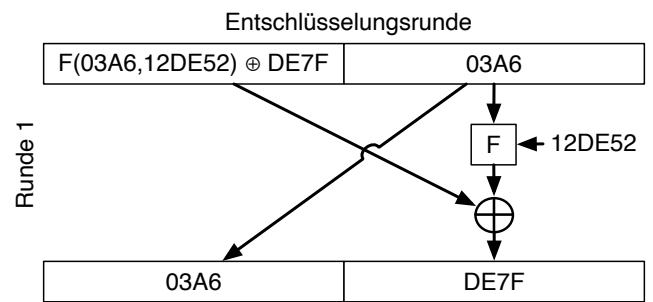
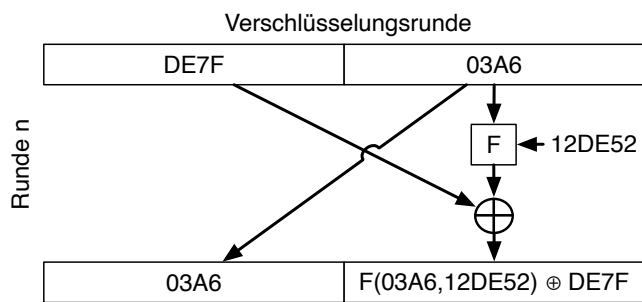
# Feistel-Chiffre

Verschlüsselung und  
Entschlüsselung



# Feistel Chiffre - Beispiel

Ver- und Entschlüsselung mit n Runden



$$[F(03A6, 12DE52) \oplus DE7F] \oplus F(03A6, 12DE52) = DE7F$$

# Feistel Chiffre - Eigenschaften

## **Rundenfunktion F:**

Größere Komplexität bedeutet in der Regel größere Resistenz gegen Kryptoanalyse.

## **Schnelle Ver-/Entschlüsselung in Software:**

Häufig ist die Verschlüsselung so in Anwendungen oder Dienstprogramme eingebettet, dass eine Hardwareimplementierung nicht möglich ist; dementsprechend ist die Geschwindigkeit des Algorithmus relevant.

## **Einfachheit der Analyse:**

Wenn der Algorithmus kurz und klar erklärt werden kann, ist es einfacher den Algorithmus auf kryptoanalytische Schwachstellen hin zu analysieren und somit ein höheres Maß an Sicherheit in Bezug auf seine Stärke zu entwickeln.

## **Algorithmus für die Ableitung der (Unter-)Schlüssel:**

Eine höhere Komplexität dieses Algorithmus sollte zu einer größeren Schwierigkeit der Kryptoanalyse führen.

## **Blockgröße:**

Größere Blockgrößen bedeuten mehr Sicherheit, aber eine geringere Verschlüsselungs-/Entschlüsselungsgeschwindigkeit für einen bestimmten Algorithmus.

## **Schlüsselgröße:**

Ein größerer Schlüssel bedeutet mehr Sicherheit, kann aber die Verschlüsselungs-/Entschlüsselungsgeschwindigkeit verringern.

## **Anzahl der Runden:**

Das Wesen der Feistel-Chiffre besteht darin, dass eine einzige Runde unzureichende Sicherheit bietet, während mehrere Runden zunehmende Sicherheit bieten.

# Data Encryption Standard (DES)

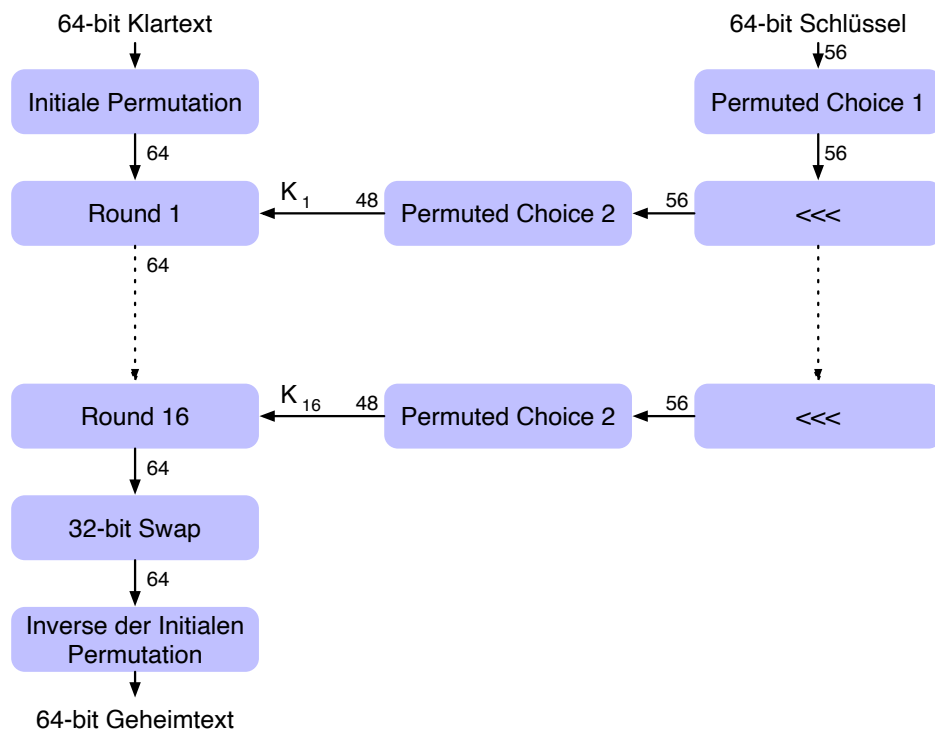
- Wurde 1977 vom National Bureau of Standards (heute NIST) als Federal Information Processing Standard 46 herausgegeben.
- War das am häufigsten verwendete Verschlüsselungsverfahren bis zur Einführung des Advanced Encryption Standard (AES) im Jahr 2001
- Der Algorithmus selbst wird als Data Encryption Algorithm (DEA) bezeichnet.

## Eigenschaften

- Die Daten werden in 64-Bit-Blöcken mit einem 56-Bit-Schlüssel verschlüsselt.
- Der Algorithmus wandelt die 64-Bit-Eingabe in einer Reihe von Schritten in eine 64-Bit-Ausgabe um.
- Dieselben Schritte werden mit demselben Schlüssel verwendet, um die Verschlüsselung rückgängig zu machen.

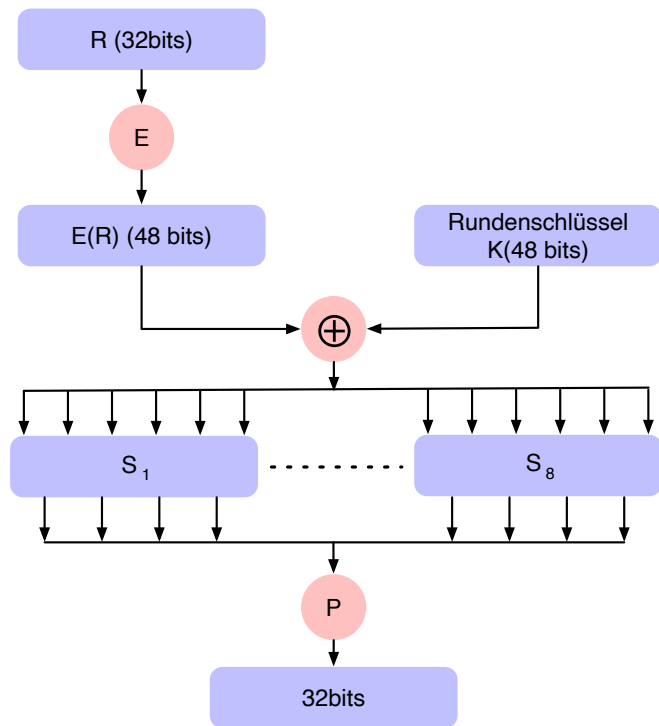
Bei DES enthält ein 64 Bit langer Schlüssel 8 Paritätsbits, die zur Überprüfung der Schlüsselübertragung verwendet werden. Die 8 Paritätsbits werden dann aus dem 64-Bit-Schlüssel entfernt. Somit ist die effektive Schlüssellänge 56 Bit.

# DES - Design





# DES Rundenfunktion („F“)



## Legende

R ist die rechte Hälfte der Nachricht.

E ist eine Expansionsfunktion.

S sind Substitutionsboxen.

P ist eine Permutation.

# DES Beispiel

| Round | Ki               | Li       | Ri       |
|-------|------------------|----------|----------|
| IP    |                  | 5a005a00 | 3cf03c0f |
| 1     | 1e030f03080d2930 | 3cf03c0f | bad22845 |
| 2     | 0a31293432242318 | bad22845 | 99e9b723 |
| 3     | 23072318201d0c1d | 99e9b723 | 0bae3b9e |
| 4     | 05261d3824311a20 | 0bae3b9e | 42415649 |
| 5     | 3325340136002025 | 42415649 | 18b3fa41 |
| 6     | 123a2d0d04262a1c | 18b3fa41 | 9616fe23 |
| 7     | 021f120b1c130611 | 9616fe23 | 67117cf2 |
| 8     | 1c10372a2832002b | 67117c12 | c11bfc09 |
| 9     | 04292a380c341103 | c11bfc09 | 887fbe6c |
| 10    | 2703212607280403 | 887fbc6c | 60017e8b |
| 11    | 2826390c31261504 | 60017e8b | f596506e |
| 12    | 12071c241a0a0108 | f596506e | 738538b8 |
| 13    | 300935393c0d100b | 73853868 | c6a62c4e |
| 14    | 311e09231321182a | c6a62c4e | 56b0bd75 |
| 15    | 283d3e0227072528 | 56b0bd75 | 75e8fd8f |
| 16    | 2921080b13143025 | 75e8fd8f | 25896490 |
| IP-1  |                  | da02ce3a | 89ecac3b |

DES-Unterschlüssel werden als acht 6-Bit-Werte im Hexadezimalformat angezeigt (der Höchstwert für  $k_i$  ist  $2^6 - 1 = 63 = 0x3F$ )

Kleine Änderung im Klartext (erster Wert +1)

| Round |                                       | δ  | Round |                                      | δ  |
|-------|---------------------------------------|----|-------|--------------------------------------|----|
|       | 02468aceeca86420<br>12468aceeca86420  | 1  | 9     | c11bfc09887fbc6c<br>996911532eed7d94 | 32 |
| 1     | 3cf03c0fbad22845<br>3cf03c0fbad32845  | 1  | 10    | 887fbc6c60017e8b<br>2eed7d94d0f23094 | 34 |
| 2     | bad2284599e9b723<br>bad3284539a9b7a3  | 5  | 11    | 600f7e8bf596506e<br>d0f23094455da9c4 | 37 |
| 3     | 99e9b7230bae3b9e<br>39a9b7a3171cb8b3  | 18 | 12    | 1596506e738538b8<br>455da9c47f6e3cf3 | 31 |
| 4     | 0bae3b9e42415649<br>171cb8b3ccaca55e  | 34 | 13    | 738538b8c6a62c4e<br>7f6e3cf34bc1a8d9 | 29 |
| 5     | 4241564918b3fa41<br>ccaca55ed16c3653  | 37 | 14    | c6a62c4e56b0bd75<br>4bc1a8d91e07d409 | 33 |
| 6     | 18b3fa419616fe23<br>d16c3653cf402c68  | 33 | 15    | 56b0bd7575e8fd81<br>1e07d4091ce2e6dc | 31 |
| 7     | 9616fe2367117cf2<br>cf402c682b2cefbcb | 32 | 16    | 75e8fd8625896490<br>1ce2e6dc365e5f59 | 32 |
| 8     | 67117cf2c11bfc09<br>2b2cefbcb99191153 | 33 | IP-1  | da02ce3a89ecac3b<br>057cde97d7683f2a | 32 |

# Lawineneffekt in DES

Kleine Änderung des Schlüssels: 0f1571c947d9e859 → 1f1571c947d9e859

| Round |                                      | δ  | Round |                                      | δ  |
|-------|--------------------------------------|----|-------|--------------------------------------|----|
|       | 02468aceeca86420<br>02468aceeca86420 | 0  | 9     | c11bfe09887fbe6c<br>548f1de471f64dfd | 34 |
| 1     | 3cf03c0fbad22845<br>3cf03c0f9ad628c5 | 3  | 10    | 8876be6c60067e8b<br>71664dfd4279876c | 36 |
| 2     | bad2284599e9b723<br>9ad628c59939136b | 11 | 11    | 60017e8bf596506e<br>4279876c399fdc0d | 32 |
| 3     | 99e9b7230bae3b9e<br>9939136676806767 | 25 | 12    | f596506e738538b8<br>399fde0d6d208dbb | 28 |
| 4     | 0bae3b9e42415649<br>768067b75a8807c5 | 29 | 13    | 738538b8c6a62c4e<br>6d208dbbb9bdeaaa | 33 |
| 5     | 4241564918b3fa41<br>5a8807c5488bde94 | 26 | 14    | c6a62c4e56b0bd75<br>b9bdeeaad2c3a56f | 30 |
| 6     | 18b3fa419616fe23<br>488dbe94aba7fe53 | 26 | 15    | 56b0bd7575e8fd8f<br>d2c3a5612765c1fb | 33 |
| 7     | 9616fe2367117cf2<br>aba7fe53177d21e4 | 27 | 16    | 75e8fd8f25896490<br>2765c1fb01263dc4 | 30 |
| 8     | 67117cf2c11bfc09<br>177d21e4548f1de4 | 32 | IP-1  | da02ce3a89ecac3b<br>ee92b50606b6260b | 30 |

## Durchschnittliche Zeit für erschöpfende Schlüsselsuche

| Schlüsselgröße<br>(bits)    | Chiffre          | Anzahl der<br>alternativen<br>Schlüssel | Zeit benötigt bei $10^9$<br>Entschlüsselungen/s | Zeit benötigt bei $10^{13}$<br>Entschlüsselungen/s |
|-----------------------------|------------------|---|---|--|
| 56                          | DES              | $2^{56} \approx 7.2 \times 10^{16}$     | 1.125 Jahre                                     | 1 Stunde   |
| 128                         | AES              | $2^{128} \approx 3.4 \times 10^{38}$    | $5.3 \times 10^{21}$ Jahre                      | $5.3 \times 10^{17}$ Jahre                         |
| 168                         | Triple DES       | $2^{168} \approx 3.7 \times 10^{50}$    | $5.8 \times 10^{33}$ Jahre                      | $5.8 \times 10^{29}$ Jahre                         |
| 192                         | AES              | $2^{192} \approx 6.3 \times 10^{57}$    | $2^{191}$ ns = $9.8 \times 10^{40}$<br>Jahre    | $9.8 \times 10^{36}$ Jahre                         |
| 256                         | AES              | $2^{256} \approx 1.2 \times 10^{77}$    | $2^{255}$ ns = $1.8 \times 10^{60}$<br>Jahre    | $1.8 \times 10^{56}$ Jahre                         |
| 26 Zeichen<br>(Permutation) | Monoalphabetisch | $26! = 4 \times 10^{26}$                | $6.3 \times 10^9$ Jahre                         | $6.3 \times 10^6$ Jahre                            |

# Stärke von DES - Timing-Angriffe

- Ein Verfahren, bei dem Informationen über den Schlüssel oder den Klartext gewonnen werden, indem beobachtet wird, wie lange eine bestimmte Implementierung für die Entschlüsselung verschiedener Chiffretexte benötigt.
- Dabei wird die Tatsache ausgenutzt, dass ein Verschlüsselungs- oder Entschlüsselungsalgorithmus für verschiedene Eingaben oft leicht unterschiedliche Zeit benötigt.
- Bislang scheint es unwahrscheinlich, dass diese Technik jemals gegen DES oder leistungsfähigere symmetrische Chiffren wie Triple DES und AES erfolgreich sein wird.

# Entwurfsprinzipien für Blockchiffre - Anzahl der Runden

- Je größer die Anzahl der Runden ist, desto schwieriger ist es, eine Kryptoanalyse durchzuführen.
- Im Allgemeinen sollte das Kriterium sein, dass die Anzahl der Runden so gewählt wird, dass bekannte kryptoanalytische Bemühungen mehr Aufwand erfordern als ein einfacher Brute-Force-Schlüsselsuchangriff.
- Hätte DES 15 oder weniger Runden, würde die differentielle Kryptoanalyse weniger Aufwand erfordern als eine Brute-Force-Schlüsselsuche.

# Entwurfsprinzipien für Blockchiffre - Funktion F

- Das Herzstück einer Feistel-Blockchiffre ist die Funktion F.
- Je nichtlinearer F ist, desto schwieriger wird jede Art von Kryptoanalyse sein.
- Der Algorithmus sollte einen großen Lawineneffekt (🇺🇸 *Avalanche-Property*) haben.

## Strict Avalanche Criterion (SAC)

Besagt, dass sich jedes Ausgangsbit  $j$  einer S-Box mit der Wahrscheinlichkeit  $1/2$  ändern sollte, wenn ein einzelnes Eingangsbit  $i$  invertiert wird und dies für alle Paare  $i, j$  gelten muss.

## Bit Independence Criterion (BIC)

Besagt, dass sich die Ausgangsbits  $j$  und  $k$  unabhängig voneinander ändern sollten, wenn ein einzelnes Eingangsbit  $i$  invertiert wird und dies für alle  $i, j$  und  $k$  gelten muss.

- Das Einhalten der SAC- und BIC-Kriterien scheint die Wirksamkeit der Verwirrungsfunktion zu stärken.



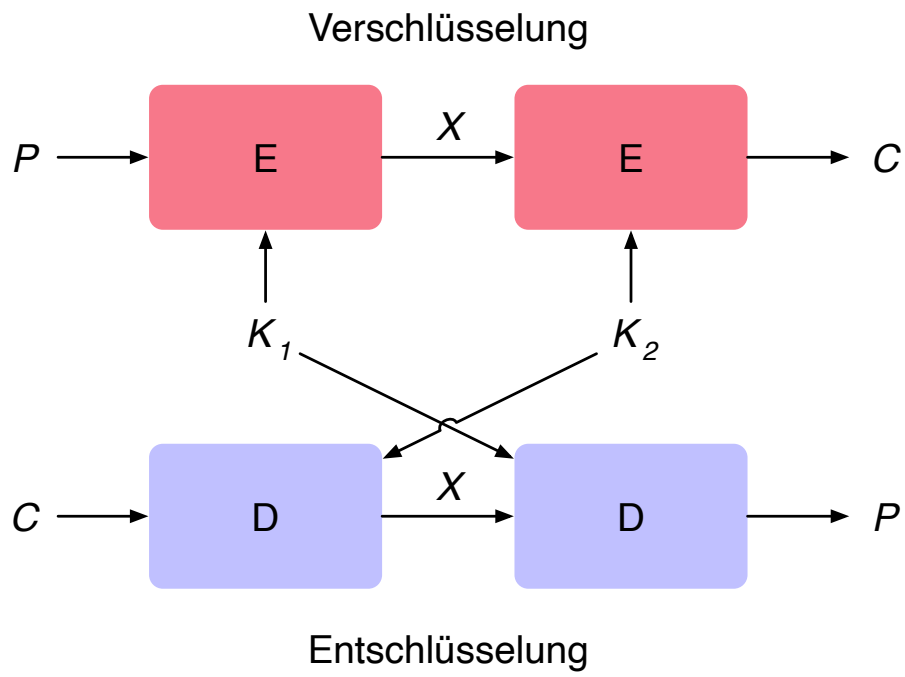
# Entwurfsprinzipien für Blockchiffre - Schlüsselableitung

- Bei jeder Feistel-Blockchiffre wird der Hauptschlüssel verwendet, um einen Unterschlüssel für jede Runde zu erzeugen.
- Im Allgemeinen möchten wir die Unterschlüssel so wählen, dass die Schwierigkeit, einzelne Unterschlüssel abzuleiten, und die Schwierigkeit, den Hauptschlüssel wieder zurückzuerhalten, maximiert werden.
- Es wird vorgeschlagen, dass die Schlüsselableitungsfunktion für die Unterschlüssel (🚩 *Key Schedule*) zumindest das **Strenge Lawinenkriterium** und das **Bit-Unabhängigkeitskriterium** für Schlüssel/Ciphertext garantieren sollte.

# 1. MEHRFACHVERSCHLÜSSELUNG

Prof. Dr. Michael Eichberg

# Doppelte Verschlüsselung



# Meet-in-the-Middle-Angriff

Beobachtung:  $E(K_2, E(K_1, P)) = E(K_3, P)$  ist nicht gültig. D. h. die zweifache Anwendung von DES führt zu einer Abbildung, die nicht äquivalent zu einer einfachen DES-Verschlüsselung ist.

1

Der Meet-in-the-Middle-Algorithmus greift dieses Verfahren an. Er hängt nicht von einer bestimmten Eigenschaft von DES ab, sondern funktioniert gegen jede Blockchiffre.

■ Möglicher Known-Plaintext-Angriff:

1. Man berechnet für alle Schlüssel  $K_1$  die Chiffretexte  $E(K_1, P)$  und speichert diese.
2. Man berechnet für alle Schlüssel  $K_2$  die Klartexte  $D(K_2, C)$ .
3. Man vergleicht die beiden Ergebnisse und prüft, ob es Übereinstimmungen gibt.

Dieser Aufwand ist lediglich doppelt so hoch wie der Aufwand bei einer einfachen Verschlüsselung.

2

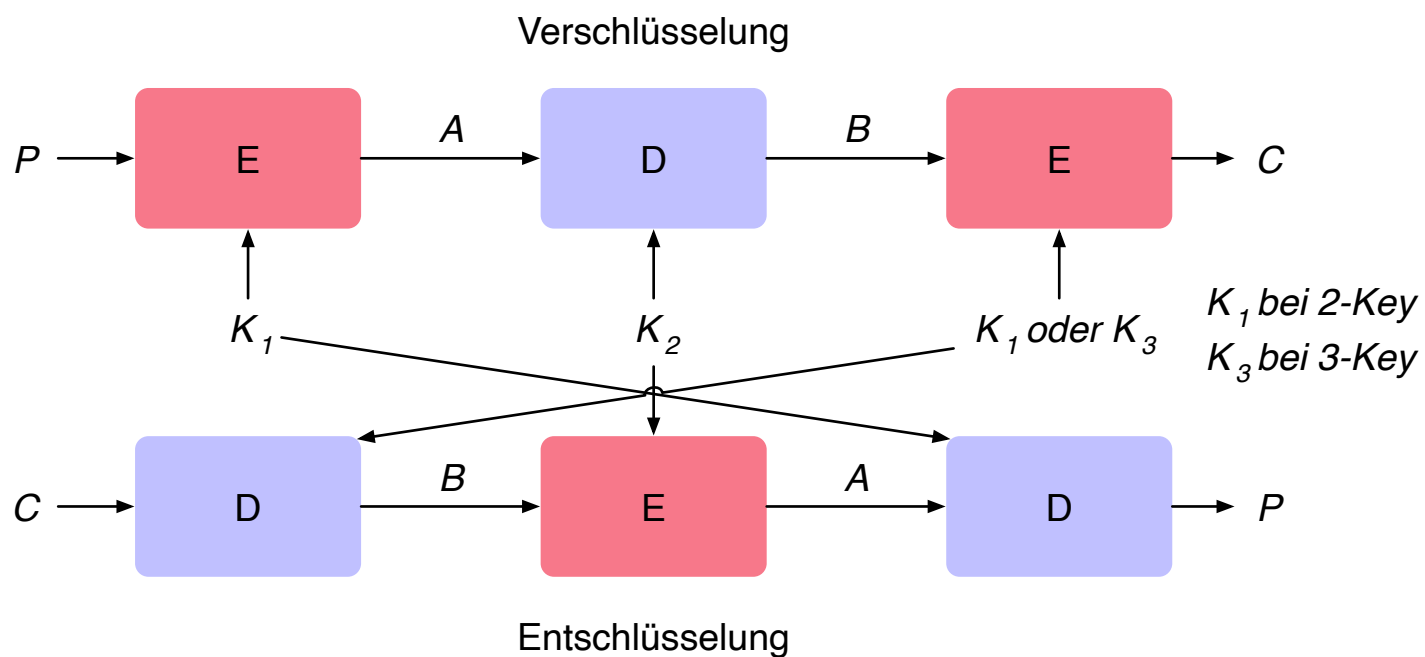
## Die zweifache Anwendung einer Blockchiffre ist nicht sinnvoll

Das Ergebnis ist, dass ein bekannter Klartextangriff gegen Doppel-DES mit einem Aufwand in der Größenordnung von  $2^{56}$  im Durchschnitt erfolgreich ist, verglichen mit durchschnittlich  $2^{55}$  für einen einfachen DES.

3

# Dreifache Verschlüsselung

(Z. B. Triple-DES (3DES) mit drei Schlüsseln)



# Triple-DES mit zwei Schlüsseln

Die offensichtliche Antwort auf den *Meet-in-the-middle*-Angriff ist die dreifache Verschlüsselung mit drei verschiedenen Schlüsseln.

- Dies erhöht die Kosten des *Meet-in-the-Middle*-Angriffs auf  $2^{112}$ , was jenseits dessen liegt, was praktikabel ist.
- Das hat den Nachteil, dass eine Schlüssellänge von  $56 \text{ bits} \times 3 = 168 \text{ bits}$  erforderlich ist, was etwas unhandlich sein kann.
- Als Alternative schlug Tuchman eine dreifache Verschlüsselungsmethode vor, die nur zwei Schlüssel verwendet.
- 3DES mit zwei Schlüsseln war eine Alternative zu DES und wurde in die Schlüsselverwaltungsstandards ANSI X9.17 und ISO 8732 aufgenommen.

# Triple-DES mit drei Schlüsseln

- Es wurden mehrere Angriffe gegen 3DES mit 2 Schlüsseln entwickelt, die jedoch (noch) nicht praktikabel sind.
- Viele Forscher sind inzwischen der Meinung, dass 3DES mit drei Schlüsseln die bevorzugte Alternative ist.
- 3DES mit drei Schlüsseln hat eine effektive Schlüssellänge von 168 Bit und ist definiert als:

$$C = E(K_3, D(K_2, E(K_1, P)))$$

- Rückwärtskompatibilität mit DES ist gegeben, wenn man  $K_3 = K_2$  oder  $K_1 = K_2$  einsetzt.

## Feistelchiffre Implementieren

Implementieren Sie eine Feistel Chiffre in einer Programmiersprache Ihrer Wahl (z.B. Java, Scala, Python, C, JavaScript ...), die es Ihnen ermöglicht:

- Nachrichten zu ver- und entschlüsseln
- Blöcke von 128 Bit zu verschlüsseln
- die Funktion  $f$  einfach auszutauschen, um die Wirkung von  $f$  zu testen
- Für die Ableitung der Rundenschlüssel können Sie eine Funktion verwenden, die eine Rotation des Schlüssels durchführt (z. B. `Integer.rotateLeft`).

### Hinweise

Kümmern Sie sich nicht um Nachrichten, die größer oder kleiner als die Blockgröße sind. Dies ist nicht notwendig, um die Auswirkungen von  $f$  oder der Verwendung eines Rundenschlüssels zu verstehen. Kümmern Sie sich nicht um einen Schlüssel, der nicht die richtige Größe hat. D.h. verwenden Sie eine Nachricht und einen Schlüssel mit der entsprechenden Größe.

Um die Austauschbarkeit der Funktion  $f$  zu erreichen können Sie je nach Sprache z.B. native Funktionen höherer Ordnung, einen Funktionszeiger oder ein Interface verwenden.



## Feistelchiffre Implementieren

Implementieren Sie eine Feistel Chiffre in einer Programmiersprache Ihrer Wahl (z.B. Java, Scala, Python, C, JavaScript ...), die es Ihnen ermöglicht:

- Nachrichten zu ver- und entschlüsseln
- Blöcke von 128 Bit zu verschlüsseln
- die Funktion  $f$  einfach auszutauschen, um die Wirkung von  $f$  zu testen
- Für die Ableitung der Rundenschlüssel können Sie eine Funktion verwenden, die eine Rotation des Schlüssels durchführt (z. B. `Integer.rotateLeft`).

## Feistelchiffre Evaluieren

1. Was passiert, wenn  $f$  nur 0x00-Werte zurückgibt (unabhängig vom Rundenschlüssel)?
2. Was passiert, wenn  $f$  nur 0x01-Werte zurückgibt (unabhängig vom Rundenschlüssel)?
3. Was passiert, wenn  $f$  einfach die entsprechende Hälfte mit dem Ergebnis der Verschiebung des Schlüssels xor?
4. Was passiert, wenn du deine Nachricht änderst? Testen Sie insbesondere, was passiert wenn die Nachricht nur aus 0x00 besteht (und Sie eine „vernünftiger“  $f$ -Funktion verwenden.)
5. Was passiert, wenn du deinen Schlüssel änderst? Was passiert in extremen Fällen (z.B. wenn das Passwort nur aus "0"s besteht?

## Feistelchiffre Evaluieren

1. Was passiert, wenn  $f$  nur 0x00-Werte zurückgibt (unabhängig vom Rundenschlüssel)?
2. Was passiert, wenn  $f$  nur 0x01-Werte zurückgibt (unabhängig vom Rundenschlüssel)?
3. Was passiert, wenn  $f$  einfach die entsprechende Hälfte mit dem Ergebnis der Verschiebung des Schlüssels xor?
4. Was passiert, wenn du deine Nachricht änderst? Testen Sie insbesondere, was passiert wenn die Nachricht nur aus 0x00 besteht (und Sie eine „vernünftiger“  $f$ -Funktion verwenden.)
5. Was passiert, wenn du deinen Schlüssel änderst? Was passiert in extremen Fällen (z.B. wenn das Passwort nur aus "0"s besteht)?