Kryptographie

Studienbrief 4

universität wien

Moderne Blockchiffren

Gerald und Susanne Teschl

Druckfehler/Feedback bitte an:
gerald.teschl@univie.ac.at

SS 23

Version:
2023-06-30

Studienbrief 4

Moderne Blockchiffren

Inhalt

Wie bereits erwähnt, unterscheidet man grundsätzlich symmetrische und asymmetrische kryptographische Verfahren. Im symmetrischen Fall haben Sender und Empfänger einen gemeinsamen geheimen Schlüssel, während im asymmetrischen Fall jeder Kommunikationspartner ein individuelles Schlüsselpaar besitzt, bestehend aus einem geheimen und einem öffentlichen Schlüssel. Wir bleiben in diesem Studienbrief bei symmetrischen Verfahren, und wenden uns nun modernen Verfahren zu. Dabei werden wir uns auf die Besprechung von **Blockchiffren** konzentrieren, also auf Verfahren, bei denen der Klartext in Blöcke zerlegt (z.B. in 128-Bit-Blöcke bei AES) und danach blockweise verschlüsselt wird.

Bei **Stromchiffren** wird der Klartext, der als Folge von Zeichen (meist Bits) vorliegt, mit einem Strom von Schlüsselzeichen verschlüsselt (z.B. mit XOR vefknüpft). Die Schlüsselfolge kann sich periodisch wiederholen (Vigenère), echt zufällig (One Time Pad) oder pseudo-zufällig sein. Der

Vorteil liegt darin, dass man nicht jedes Mal mit der Ver- bzw. Entschlüsselung warten muss, bis ein vollständiger Block vorliegt. Wie man Blockchiffren an verschiedene Anforderungen anpassen (und z.B. eine Stromchiffre erhalten kann), werden wir in Abschnitt 4.4 besprechen.

Für einen Überblick über die Verwendung von modernen Blockchiffren siehe zum Beispiel [NIST17]. Demnach wird durch das NIST (US National Institute of Standards and Technology) aktuell die Blockchiffre AES zur Ver- bzw. Entschlüsselung empfohlen (*approved*). Weitere Empfehlungen finden Sie bei der European Union Agency for Network and Information Security (enisa) [ENISA] und beim deutschen Bundesamt für Sicherheit in der Informationstechnik (BSI) [BSI]. Zunächst folgt nun eine Besprechung des DES, der 1977 als US-Verschlüsselungsstandard veröffentlicht wurde (*Federal Information Processing Standard* [FIPS46-3]). Im Jahr 2005 wurde er wegen seiner inzwischen zu geringen Schlüssellänge zurückgezogen.

4.1 Data Encryption Standard (DES)

In Abschnitt 2.4 haben wir gesehen, dass auch die auf den ersten Blick kompliziert und daher sicher wirkende Vigenère-Chiffre gebrochen werden kann. Überlegen wir, was die Schwachstellen der Vigenère-Chiffre sind:

Ein Problem ist die Tatsache, dass jeder Buchstabe für sich verschlüsselt wird. Insbesondere bedeutet das, dass eine Änderung eines Klartextbuchstaben nur die Änderung eines einzigen Geheimtextbuchstaben bewirkt. Ein guter Verschlüsselungsalgorithmus sollte aber die im Klartext enthaltene Information möglichst breit „verschmieren“. Claude Shannon (1916–2001), der Begründer der Informationstheorie, hat dafür den Begriff **Diffusion** (*diffusion*) eingeführt [Sha49]. Insbesondere soll die Information jedes einzelnen Klartextbits alle Geheimtextbits beeinflussen (**Vollständigkeit**) und die Änderung eines beliebigen Klartextbits soll jedes Geheimtextbit mit Wahrscheinlichkeit $\frac{1}{2}$ ändern (**Lawineneffekt** bzw. **Avalanche Effect**).

Weiters bietet die Vigenère-Chiffre keinerlei Sicherheit gegenüber einem Klartextangriff. In der Praxis ist es nämlich so gut wie unvermeidbar, dass zu einigen Geheimtextbuchstaben auch die zugehörigen Klartextbuchstaben bekannt sind (z.B. ist das Dateiformat bekannt und damit die ersten Klartext-Bytes). Für ein solches Paar aus zusammengehörendem Klartext/Geheimtextbuchstaben lässt sich bei der Vigenère-Chiffre sofort der zugehörige Schlüsselbuchstabe ermitteln, da dazu nur eine einfache Gleichung gelöst werden muss: aus $y_i = x_i + e_{i \bmod k} (\mathrm{mod}\, n)$ folgt $e_{i \bmod k} = y_i - x_i (\mathrm{mod}\, n)$. Jeder bekannte Schlüsselbuchstabe bedeutet hier, dass damit ein Teil des Geheimtextes entschlüsselt werden kann, und dass sich – für einen Brute-Force-Angriff – die Anzahl der zu probierenden Schlüssel vermindert. Eine ähnliche Situation besteht auch beim Angriff auf eine Hill-Chiffre (siehe Übung 6 in Studienbrief 2), wo der Zusammenhang zwischen Klartext und Geheimtext durch ein lineares Gleichungssystem gegeben ist. Wenn bei einer Hill-Chiffre daher ein

ausreichend langes zusammengehörendes Klartext-/Geheimtextstück bekannt ist, dann kann leicht nach dem Schlüssel aufgelöst und somit der gesamte Schlüssel berechnet werden. In der Tat gilt dies für jedes *lineare* Verschlüsselungsverfahren: Da eine lineare Abbildung ja bekanntlich durch die Bilder der Basisvektoren eindeutig bestimmt ist, benötigt man bei n-Bit Blöcken nur n linear unabhängige Klartextblöcke zusammen mit ihren zugehörigen Geheimtextblöcken, um die Entschlüsselungsfunktion zu berechnen (bei einer *affinen* Abbildung braucht man noch um ein geeignetes Paar mehr). Bei einem guten Verschlüsselungsalgorithmus müssen also *nichtlineare* Funktionen verwendet werden. Shannon hat die Forderung, dass der Zusammenhang zwischen Klartext, Geheimtext und geheimem Schlüssel möglichst „kompliziert“ sein soll, als **Konfusion** (*confusion*) bezeichnet.

Ein Verschlüsselungsverfahren, das die oben genannten Forderungen erfüllt, ist der **Data Encryption Standard (DES)**. Im Jahre 1977 hat das National Institute of Standards and Technology (NIST) diesen von der Firma IBM entwickelten Algorithmus als offiziellen Verschlüsselungsstandard in den USA festgelegt. DES wurde nach einer öffentlichen Ausschreibung aus mehreren Vorschlägen ausgewählt. Er wurde zuvor und auch danach weltweit gründlichen, öffentlich zugänglichen Prüfungen durch Expert:innen unterzogen (Kerckhoffs'sche Prinzip). Aus diesem Grund sind auch bis heute noch keine erfolgreichen Angriffe (außer Brute-Force) gegen ihn bekannt. Heute ist er wegen seiner zu geringen Schlüssellänge nicht mehr resistent gegen Brute-Force-Angriffe und somit nicht mehr sicher. Wir besprechen DES als klassisches Beispiel einer modernen Blockchiffre, da viele Ideen/Begriffe weiterhin von Bedeutung sind.

Der DES ist eine **Blockchiffre**, die auf 64-Bit Blöcken operiert. Das heißt, der Klartext (als Folge von Bits) wird in Blöcke der Länge 64 Bit zerlegt (der letzte Block muss eventuell aufgefüllt werden, was man als *padding* bezeichnet) und danach blockweise verschlüsselt. Der Geheimtext besteht dann wieder aus einer Folge von 64 Bit Blöcken. Es gibt 2^{64} mögliche Klartextblöcke (und ebenso viele Geheimtextblöcke). Man kann den DES daher auch als ein monoalphabetisches Verfahren auf einem Alphabet von 2^{64} Buchstaben (1 Block = 1 Buchstabe) auffassen. Aufgrund der großen Anzahl von möglichen Blöcken sind die statistischen Angriffe aus Studienbrief 2 aber nicht möglich, da dazu ein viel zu langer Klartext vorliegen müsste (damit die einzelnen Blöcke für statistische Aussagen hinreichend oft im Text vorkommen).

Ein geheimer Schlüssel bei DES ist 64 Bit lang, jedes 8te Bit ist allerdings ein Paritätsbit (also ein Prüfbit), wodurch eine effektive Schlüssellänge von 56 Bit vorliegt (aufgrund einer Symmetrie – siehe Aufgabe 5 – sind es sogar nur 55). Für einen Brute-Force-Angriff müssten also 2^{56} mögliche Schlüssel durchprobiert werden. Im Jahr 1977 war das noch eine ausreichend große Hürde.

Sehen wir uns nun den DES im Detail an: Aufgrund der eingangs erwähnten Forderungen (Diffusion, Nichtlinearität) ist es nicht verwunderlich, dass er nicht durch eine einfache Vorschrift gegeben ist: DES setzt sich aus mehreren Operatio-

nen zusammen, die hintereinander und in mehreren Runden ausgeführt werden. Man spricht deshalb von einer **Produktchiffre**. Wir sehen uns die „Bauteile“ im Folgenden einzeln an: Die Operationen bestehen im Wesentlichen aus Permutationen (= Vertauschung der Reihenfolge) der Bits, XOR-Verknüpfung mit dem geheimen Schlüssel und aus sogenannten **Feistel-Boxen**. Feistel-Boxen sind auch Bestandteil anderer, moderner Algorithmen (z.B. MARS, Twofish). Man spricht allgemein von **Feistel-Netzwerken**.

Der Name geht zurück auf den IBM-Mitarbeiter Horst Feistel (1915–1990), der die Grundlage des DES, den sogenannten LUCIFER-Algorithmus, mitentwickelt hat.

Feistel-Box $F_k(l, r)$

Die Feistel-Boxen enthalten den nichtlinearen Bestandteil des DES und sind somit ein Kernstück für die Sicherheit des Algorithmus. Eine Feistel-Box führt beim DES 64 Eingangsbits in 64 Ausgangsbits über. Um ihre Bedeutung richtig einschätzen zu können, muss man sich zunächst klarmachen, dass die Eingangsbits nicht einfach irgendwie planlos durchgemischt werden können. Denn der Empfänger der Nachricht muss das Chaos, das angerichtet wird, ja wieder rückgängig machen (sprich entschlüsseln) können. Es muss also möglich sein, die nichtlineare Abbildung zu invertieren.

Das Invertieren bei einer *linearen* Abbildung ist einfach, denn man braucht nur ein lineares Gleichungssystems zu lösen (anhand einer Gleichung illustriert: Auflösung von $y = 5x + 3$ nach x ist kein Problem). Bei einem *nichtlinearen* Gleichungssystem (siehe z.B. $y = 4x^5 + 2x + 3$) ist das Auflösen nach x erstens viel schwieriger und zweitens gibt es im Allgemeinen keine eindeutige Lösung.

Die Lösung des Invertierproblems ist beim DES die **Feistel-Box**: Man zerlegt die 64 Eingangsbits $x_1x_2 \dots x_{64}$ in zwei 32-Bit-Blöcke $l = x_1x_2 \dots x_{32}$ und $r = x_{33} \dots x_{64}$ (l bzw. r steht für „links“ bzw. „rechts“) und definiert die **Feistel-Funktion**

$$F_k(l, r) = (l \oplus f(r, k), r).$$

(Hier ist k der geheime Teilschlüssel der jeweiligen Runde.) Die darin vorkommende Funktion f besprechen wir später. Sie ist beim DES eine bestimmte, nichtlineare Funktion, die in jeder Runde gemeinsam mit dem Rundenschlüssel k verwendet wird. Die zentrale Eigenschaft der Feistelfunktion ist, dass sie invertierbar ist, auch wenn f nicht invertierbar ist. Wegen $x \oplus y \oplus y = x$ gilt nämlich: $F_k(F_k(l, r)) = F_k(l \oplus f(r, k), r) = (l \oplus f(r, k) \oplus f(r, k), r) = (l, r)$. Die Abbildung F_k ist also ihr eigenes Inverses: $F_k^{-1} = F_k$. Bei der Wahl von f stehen also alle Möglichkeiten offen.

Vertauschung von „Links“ und „Rechts“

Wie man sieht, wird der rechte 32-Bit-Block des Eingangs durch die Feistel-Box nicht verändert. Damit im Folgenden auch diese Bits verändert werden, werden nach der Feistel-Box der linke und der rechte 32-Bit-Block vertauscht:

$$\Theta(l, r) = (r, l).$$

Es folgt die zweite Feistel-Box, danach wieder Vertauschung von „Links“ und „Rechts“, danach die dritte Feistel-Box usw. Insgesamt gibt es beim DES 16 Run-

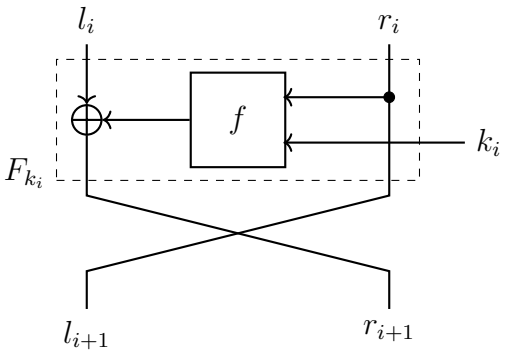

Abbildung 4.1: Eine Runde mit Nummer i in einem Feistel-Netzwerk: Die linke Hälfte l_i der Eingangsbits wird in der Feistel-Box F_{k_i} durch die nichtlineare Funktion f unter Verwendung des Rundenschlüssels k_i und der rechten Hälfte r_i der Eingangsbits verschlüsselt; die rechte Hälfte r_i wird durch die Feistel-Box nicht verändert.

den bestehend aus jeweils einer Feistel-Box und einer Vertauschung wie in Abbildung 4.1 veranschaulicht. In jeder Runde wird dabei ein eigener Rundenschlüssel verwendet.

Teilschlüsselerzeugung

Die 16 **Rundenschlüssel** (**Teilschlüssel**) zu je 48 Bits werden aus dem geheimen, 64 Bit langen, DES-Schlüssel (**Hauptschlüssel**) erzeugt. Dies geschieht nach einem festen „Fahrplan“ (*key schedule*), der vorgibt, welche Bits des Hauptschlüssels in welcher Runde ausgewählt und in welcher Reihenfolge sie zum Teilschlüssel zusammengefügt werden. Zu Beginn werden dabei aus dem 64-Bit DES-Schlüssel die acht Kontrollbits entfernt und die verbleibenden 56 Bits permutiert (Permuted Choice 1). Danach gibt es zwei Elemente, die einander abwechseln (für Details siehe [FIPS46-3]):

- **Left Shifts**: Die 56 Schlüsselbits werden geteilt und jede Hälfte wird zyklisch nach links verschoben. (Zyklische Verschiebung einer Folge um eine

Stelle nach links kann man sich so vorstellen, dass man die Folge nach links schiebt, und das „herabfallende“ Element rechts wieder anfügt.) Die Anzahl der Stellen, um die dabei verschoben wird, hängt von der Rundennummer ab.

- **Permuted Choice 2**: Linke und rechte Hälfte werden zusammengefügt und es werden 48 Bits für den jeweiligen Teilschlüssel ausgewählt und angeordnet.

Eingangs-/Ausgangspermutation

Der letzte Bestandteil (ohne kryptographische Bedeutung) des DES ist eine fest vorgegebene Permutation IP der 64-Bits (*initial permutation*, siehe [FIPS46-3])

$$IP(x_1x_2x_3\ldots x_{64}) = x_{58}x_{50}x_{42}\ldots x_7$$

und ihre Umkehrung

$$IP^{-1}(x_1x_2\ldots x_{64}) = x_{40}x_8\ldots x_{25}.$$

Nachdem wir die Bestandteile des DES jetzt einzeln besprochen haben, können wir nun seine prinzipielle Struktur anschreiben:

$$y = DES_k(x) = (IP^{-1} \circ F_{k_{16}} \circ \Theta \circ \ldots \circ F_{k_2} \circ \Theta \circ F_{k_1} \circ IP)(x)$$

Nach der Eingangspermutation IP durchläuft ein Klartextblock x insgesamt 16 **Runden**, von denen jede gleich aufgebaut ist (Feistel-Box F_{k_i} mit eigenem Rundenschlüssel k_i und nachfolgende Vertauschung Θ). Den Abschluss bildet die Ausgangspermutation IP^{-1}.

Da, wie besprochen, eine Feistel-Box F_{k_i} ihre eigene Inverse ist, und das natürlich auch für die Vertauschung Θ gilt, ist der Entschlüsselungsalgorithmus DES^{-1} gleich wie der Verschlüsselungsalgorithmus. Es müssen nur die Teilschlüssel in umgekehrter Reihenfolge verwendet werden:

$$x = DES_k^{-1}(y) = (IP^{-1} \circ F_{k_1} \circ \Theta \circ \ldots \circ \Theta \circ F_{k_{16}} \circ IP)(y)$$

Nun sehen wir uns noch die nichtlineare **f-Box** $f(r, k_i)$, die in den Feistel-Boxen verwendet wird, genauer an. Es wurde ja schon die wichtige Tatsache erwähnt, dass man beim Design von f auf *Invertierbarkeit* keine Rücksicht zu nehmen brauchte, weil diese dank der speziellen Konstruktion der Feistel-Box unabhängig von f gesichert ist. Daher hatten IBM bzw. das NIST jegliche Freiheit bei der Wahl von f und diese Funktion wurde in der Tat so geschickt gewählt, dass auch Jahrzehnte danach keine erfolgreichen mathematischen Angriffe auf DES bekannt sind. Grundsätzlich bildet f den 32-Bit-Block r_i unter Verwendung des Teilschlüssels k_i auf einen 32-Bit Block $f(r_i, k_i)$ ab. Hier ein grober Überblick über die einzelnen Elemente von f:

- **Expansionspermutation (E-Box)**: Die 32 Eingangsbits r_i werden permutiert und auf 48 Bits aufgestockt („*expansion*“), indem gewisse Bits des Inputs doppelt verwendet werden.

- **XOR mit dem Teilschlüssel**: Die nunmehrigen 48 Bits werden mit dem 48-Bit Teilschlüssel k_i XOR-verknüpft.

- **Substitutionstabellen (S-Box)**: Nun folgt die eigentliche nichtlineare Abbildung, die zwar mathematisch angegeben werden kann, aber effektiver mithilfe von Wertetabellen implementiert wird. Der 48-Bit-Block wird in acht Blöcke zu je 6 Bits aufgeteilt. Jeder dieser acht 6-Bit-Blöcke wird mittels Wertetabelle (S_1, S_2, …, S_8) auf einen 4-Bit-Block abgebildet. Danach werden die acht entstandenen 4-Bit-Blöcke wieder zu 32-Bit zusammengefügt.

- **Permutation (P-Box)**: Zuletzt wird der 32-Bit-Block permutiert, also die Reihenfolge der Bits geändert.

Der Aufbau ist in Abbildung 4.2 veranschaulicht. Die Wertetabellen für die E-Box, die P-Box und die S-Boxen sind in [FIPS46-3] angegeben.

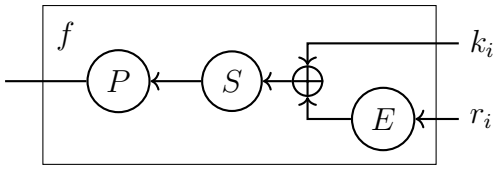

Abbildung 4.2: Schematischer Aufbau der Rundenfunktion f beim DES

Sehen wir uns nun abschließend noch einmal das Design von DES an und machen wir uns bewusst, welche Aufgabe jeder Teil besitzt:

- Die Eingangs- und Ausgangspermutation IP bzw. IP^{-1} haben als einzige keinerlei kryptographische Bedeutung und wahrscheinlich historische Wurzeln im ursprünglichen Hardwaredesign.

- Die Expansionspermutation E am Eingang der f-Funktion sorgt dafür, dass eine Änderung gewisser Klartextbits nicht nur ein, sondern zwei Geheimtextbits beeinflusst. Zudem sorgt die Permutation P am Ausgang der f-Funktion, dass die Ausgangsbits einer S-Box in der nächsten Runde mehrere S-Boxen durchlaufen. Im Verlauf der 16 Runden kommt es daher zum geforderten Lawineneffekt (Diffusion).

- Die Substitution mittels Wertetabellen S_i in der f-Funktion ist der nichtlineare Bestandteil des Algorithmus (Konfusion).

- Die zyklische Permutation und Auswahl der Bits bei der Teilschlüsselerzeugung stellen sicher, dass möglichst jedes Bit des Hauptschlüssels alle Geheimtextbits beeinflusst. In jeder Runde kommen andere Schlüsselbits zum Einsatz, und zwar jedes Bit etwa in 14 Runden. Dies allerdings wieder nicht ganz gleichmäßig, denn diese Tatsache könnte sonst bei der Kryptoanalyse genutzt werden.

DES wurde auf eine einfache Implementierung in Hardware optimiert: Es werden nur Permutationen, Substitutionen und XOR verwendet, keinerlei Additionen oder Multiplikationen (die mehr Rechenaufwand bedeuten würden). Dass der nichtlineare Anteil aus acht kleinen S-Boxen besteht, liegt daran, dass der Speicherbedarf für die Substitutionstabelle einer einzigen großen S-Box zu groß wäre. Denn die 8 kleinen S-Boxen von 2^6 auf 2^4 Bit benötigen lediglich 512 Bytes ($8 \cdot 2^6 * 2^4$ Bit), während eine große S-Box von 2^{48} auf 2^{32} Bit mit 1/8 Yotabyte ($2^{48} \cdot 2^{32}$ Bit) zu Buche schlagen würde.

Sicherheit

Seit der Einführung des DES im Jahr 1977 versuchten Wissenschaftler:innen weltweit, diesen zu brechen. Im Jahr 1990 wurde von den israelischen Kryptologen Eli Biham (*1960) und Adi Shamir (*1952) die **Differentielle Kryptoanalyse** entwickelt [BS91], [BS93]. Dabei handelt es sich um einen Chosen-Plaintext-Angriff, der durch die Wahl von (in der Regel sehr vielen) Klartextpaaren, die sich um eine feste Differenz unterscheiden, in der Lage ist Teile des Schlüssels zu finden. Die Anzahl der sechzehn Runden ist dabei essentiell; wird sie reduziert, so ist ein solcher Angriff praktikabel.

Weiters war beim Entwurf des nichtlinearen Anteils (der S-Boxen) zu beachten, dass es keine lineare Beziehungen gibt, die diese Bestandteile gut beschreiben. Sonst kann man mittels der von Mitsuru Matsui (*1961) [Mat93] entwickelten **Linearen Kryptoanalyse** Rückschlüsse auf den Schlüssel ziehen. Dabei handelt es sich um einen Klartext-Angriff der ebenfalls viele Klar-/Geheimtextpaare benötigt. Wir werden auf diese Verfahren im nächsten Abschnitt noch genauer eingehen.

DES wurde zwar von IBM entwickelt, nach der Begutachtung durch den US-Geheimdienst NSA (National Security Agency) wurden aber angeblich von dieser alle S-Boxen geändert. Da diesbezügliche Einzelheiten natürlich geheim sind, wurde immer wieder spekuliert, ob die NSA bei DES nicht doch eine Hintertür eingebaut hat, die es ihr ermöglicht, mit DES verschlüsselte Nachrichten zu entschlüsseln. Vielleicht wollte die NSA aber auch nur sicherstellen, dass IBM keine solche Hintertür eingebaut hat. Don Coppersmith, ein ehemaliger Mitarbeiter von IBM und Mitentwickler des DES, veröffentlichte im Jahr 1994 die Designkriterien für den DES [Cop94] und erklärte:
The entire algorithm was published [...], but the design considerations, which we present here, were not published at that time. The design took advantage of knowledge of certain cryptanalytic techniques, most prominently the technique of „differential cryptanalys“, which were not known in the published literature. After discussions with NSA, it was decided that disclosure of the design considerations would reveal the technique of differential cryptanalysis, a powerful technique that can be used against many ciphers. This in turn would weaken the competitive advantage the United States enjoyed over other countries in the field of cryptography. Many people speculated,

however, that the lack of disclosure was due to some "trap door" or hidden weakness in the DES. One of the purposes of the present paper is to dispel this notion and to indicate that, in fact, the reason for not publishing the criteria lay in the hidden strengths of the algorithm, not hidden weaknesses.

Bis heute gibt es keine bekannte Schwachstelle von DES, dennoch ist aber die Rechenleistung moderner Computer seit Entwicklung von DES so weit gestiegen, dass (aufgrund der mittlerweile zu kleinen Schlüssellänge) ein Brute-Force-Angriff heute kein Problem mehr darstellt. Um die Behauptungen der damaligen US-Regierung zur Sicherheit von DES zu widerlegen, wurde im Jahre 1998 von der **Electronic Frontier Foundation** (EFF, eine US-Nichtregierungsorganisation zum Schutz von digitalen Grundrechten) mit einem Budget von 250 000$ ein Prototyp eines Spezialchip namens „**Deep Crack**" entwickelt, der DES innerhalb von wenigen Tagen entschlüsselt.

DES wurde im Jahr 1999 als Standard zwar nochmals vom NIST bestätigt [FIPS46-3], jedoch wurde seine Anwendung als **Triple DES oder TDEA (Triple Data Encryption Algorithm)** empfohlen [TDEA17]: Es sind dazu drei (unabhängige) DES-Schlüssel k_1, k_2, k_3 (*key bundle*) zu wählen. Der Klartextblock x wird dann zunächst mit dem Schlüssel k_1 verschlüsselt, dann mit k_2 „entschlüsselt", und zuletzt wieder mit k_3 verschlüsselt:

$$y = \mathrm{TDEA}(x) = (DES_{k_3} \circ DES_{k_2}^{-1} \circ DES_{k_1})(x).$$

Durch die Verwendung von drei (unabhängigen) Schlüsseln wird die Schlüssellänge allerdings effektiv nur verdoppelt, beträgt also 112 Bit, wie wir gleich sehen werden. Ein Brute-Force-Angriff ist damit aber praktisch nicht mehr erfolgreich durchführbar.

Die Verwendung von DES^{-1} im mittleren Schritt (also *Ent*schlüsselung mit k_2 anstelle von *Ver*schlüsselung) hat nur Kompatibilitätsgründe. Denn dadurch kann man, wenn man nur den „einfachen" DES verwenden möchte (um z.B. in der Vergangenheit verschlüsselte Dokumente zu entschlüsseln), einfach $k_1 = k_2 = k_3$ wählen.

Warum ist die effektive Schlüssellänge nur 112 und nicht $3 \cdot 56 = 168$ Bits? Das liegt daran, dass folgender Klartextangriff möglich ist (**Meet-in-the-Middle-Angriff**): Angenommen, Mallory besitzt ein zusammengehörendes Klartext-Geheimtext-Paar x, y. Dann weiß er:

$$y = (DES_{k_3} \circ DES_{k_2}^{-1} \circ DES_{k_1})(x),$$

oder äquivalent dazu (auf beiden Seiten $DES_{k_3}^{-1}$ anwenden)

$$DES_{k_3}^{-1}(y) = (DES_{k_2}^{-1} \circ DES_{k_1})(x).$$

Nun berechnet Mallory für alle k_3 die Werte $DES_{k_3}^{-1}(y)$ und speichert diese; danach berechnet er für alle (k_2, k_1) die Werte $(DES_{k_2}^{-1} \circ DES_{k_1})(x)$ und vergleicht mit den gespeicherten Werten, bis Übereinstimmung auftritt. Der Aufwand dafür ist etwa

$2^{56} + 2^{112} \approx 2^{112}$, entspricht also einer effektiven Schlüssellänge von 112 Bit. Siehe hierzu auch z.B. die Bemerkungen in [Bar20].

Dieser Angriff benötigt allerdings einiges an Speicher (Aufgabe 7), es gibt aber auch Möglichkeiten den Speicheraufwand auf Kosten des Rechenaufwands zu reduzieren.

Eine zweifache Hintereinanderausführung von DES kann auch auf die eben beschriebene Art angegriffen werden und bringt somit keinen signifikant besseren Schutz als einfache Verschlüsselung mit DES. Deshalb gibt es keinen „Double DES“. Ein Meet-in-the-Middle-Angriff ist übrigens immer anwendbar, wenn ein Algorithmus aus der Hintereinanderausführung zweier Teile besteht, in die jeweils nur ein Teil des Schlüssels eingeht.

DES wurde 2005 vom NIST endgültig außer Kraft gesetzt und soll heute nur noch in Form von TDEA verwendet werden. Bereits im Jahr 2001 wurde DES durch den AES als neuer Standard abgelöst. Auch TDEA hat inzwischen ein Ablaufdatum. Er soll nur noch bis Dezember 2023 zur Verschlüsselung verwendet werden (siehe [BR19], Seite 7).

4.1.1 S-DES

Im Prinzip kann man die Details zu den in DES verwendeten Operationen im bereits zitierten Standard [FIPS46-3] nachlesen. Aufgrund der Länge der Ausdrücke ist das zeitweise recht mühsam und wir besprechen deshalb hier eine weit verbreitete vereinfachte Variante, bekannt als **S-DES** (*Simplifyed DES*, auch **Baby-DES**) [Sch96], für didaktische Zwecke. Trotz seiner Einfachheit (und Unsicherheit) kann man mit seiner Hilfe wesentlichen Charakteristika des DES mit geringerem Aufwand nachvollziehen. S-DES operiert auf 8-Bit-Blöcken (statt 64 bei DES), durchläuft nur zwei Runden (statt 16) und der geheime Hauptschlüssel ist 10 Bit lang (statt 56).

Die prinzipielle Struktur des S-DES lautet

$$y = SDES_k(x) = (IP^{-1} \circ F_{k^2} \circ \Theta \circ F_{k^1} \circ IP)(x),$$

wobei $x = x_1x_2x_3x_4x_5x_6x_7x_8$ einen 8-Bit Klartextblock und $y = y_1y_2 \dots y_8$ den zugehörigen 8-Bit Geheimtextblock bezeichnet. Zunächst ist

$$IP(x_1x_2x_3x_4x_5x_6x_7x_8) = x_2x_6x_3x_1x_4x_8x_5x_7$$

eine fix vorgegebene Permutation der Klartextbits (**Eingangspermutation**), und IP^{-1} die zugehörige Umkehrabbildung (**Ausgangspermutation**):

$$IP^{-1}(x_1x_2x_3x_4x_5x_6x_7x_8) = x_4x_1x_3x_5x_7x_2x_8x_6$$

Entschlüsselt wird, indem die inversen Abbildungen in umgekehrter Reihenfolge angewendet werden, also mit

$$x = SDES_k^{-1}(y) = (IP^{-1} \circ F_{k^1} \circ \Theta \circ F_{k^2} \circ IP)(y).$$

Für jede Runde wird aus dem geheimen Hauptschlüssel $k = k_1k_2 \dots k_{10}$ einen 8-Bit Teilschlüssel erzeugt (**Teilschlüsselerzeugung**): Zunächst werden die 10 Bit des Schlüssels permutiert

$$P_{key}(k_1k_2k_3k_4k_5\, k_6k_7k_8k_9k_{10}) = k_3k_5k_2k_7k_4\, k_{10}k_1k_9k_8k_6$$

und danach in zwei 5-Bit-Blöcke $k_3k_5k_2k_7k_4$ und $k_{10}k_1k_9k_8k_6$ zerlegt. Dann wird jeder der Blöcke um eine Stelle zyklisch nach links verschoben, also $k_5k_2k_7k_4k_3$ bzw. $k_1k_9k_8k_6k_{10}$. Diese beiden neuen Hälften werden wieder zu 10 Bit aneinandergefügt, es ergibt sich also $k_5k_2k_7k_4k_3\, k_1k_9k_8k_6k_{10}$, und daraus werden 8 Bits ausgewählt (die Bits an den Positionen $6, 3, 7, 4, 8, 5, 10, 9$ – **Kompressionspermutation**), die den ersten Teilschlüssel 1 bilden: $k^1 = k_1k_7k_9k_4\, k_8k_3k_{10}k_6$.

Nun trennt man die 10 Bit $k_5k_2k_7k_4k_3\, k_1k_9k_8k_6k_{10}$ wieder und verschiebt jede Hälfte um zwei Stellen zyklisch nach links weiter. Wie zuvor werden wieder 8 Bits ausgewählt und ergeben den zweiten Teilschlüssel: $k^2 = k_8k_3k_6k_5\, k_{10}k_2k_9k_1$.

Beispiel 4.1 Teilschlüsselerzeugung
Für den Schlüssel $(34)_{10} = (00001\,00010)_2$ erhält man die beiden Teilschlüssel $k^1 = 0010\,0000$ und $k^2 = 0001\,0010$.

Die f-Box $f(x, k)$: Die Abbildung operiert auf 4-Bit $x_1x_2x_3x_4$, die wir zunächst permutieren, zu 8-Bit expandieren (**Expansionspermutation**)

$$P_E(x_1x_2x_3x_4) = x_4x_1x_2x_3\, x_2x_3x_4x_1$$

und dann mit dem Teilschlüssel k verknüpfen:

$$y_1 \dots y_8 = x_4x_1x_2x_3\, x_2x_3x_4x_1 \oplus k_1k_2k_3k_4\, k_5k_6k_7k_8,$$

wobei k_i das i-te Bit des Teilschlüssels ist. Nun fassen wir die Bits zu Zweierblöcken zusammen: y_1y_4, y_2y_3, y_5y_8, y_6y_7, und wandeln jeden Zweierblock in eine Dezimalzahl zwischen 0 und 3 um, zu der wir eins addieren. Das ergibt vier Dezimalzahlen q_1, q_2, q_3, q_4 zwischen 1 und 4. Nun nehmen wir den Eintrag q_1, q_2 der Matrix S^1 (d.h., Zeile q_1, Spalte q_2) und den Eintrag q_3, q_4 der Matrix S^2. Diese Matrizen sind fix vorgegeben:

$$S^1 = \begin{pmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{pmatrix}, \qquad S^2 = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{pmatrix}.$$

Diese Einträge wandeln wir nun wieder in zwei 2-Bit-Blöcke x_1x_2 und x_3x_4 um, die zusammen vier Bit $x_1x_2x_3x_4$ ergeben, und nach einer weiteren Permutation

$$P_f(x_1x_2x_3x_4) = x_2x_4x_3x_1$$

die Ausgabe von f bilden. Die f-Box besteht also aus vier Teilen, der Expansionspermutation (E-Box), der Schlüsseladdition, der nichtlinearen Abbildung mittels der zwei Matrizen (S-Box) und der Permutation (P-Box).

Beispiel 4.2 **f-Box**
Berechnen Sie f für
a) $x = 0100$ mit dem Teilschlüssel $k^1 = 0010\,0000$
b) $x = 1110$ mit dem Teilschlüssel $k^2 = 0001\,0010$.

Lösung zu 4.2

a) Die Expansionspermutation macht aus den gegebenen $x_1x_2x_3x_4 = 0100$ den 8-Bit-Block $x_4x_1x_2x_3\,x_2x_3x_4x_1 = 0010\,1000$, der mit k^1 verknüpft wird:

$$y_1 \dots y_8 = 0010\,1000 \oplus 0010\,0000 = 0000\,1000.$$

Dieser String wird in Zweierblöcke $y_1y_4 = 00$, $y_2y_3 = 00$, $y_5y_8 = 10$, $y_6y_7 = 00$ aufgeteilt, diese in die Dezimalzahlen $0, 0, 2, 0$ umgerechnet; jeweils 1 addiert liefert die Dezimalzahlen $q_1, q_2, q_3, q_4 = 1, 1, 3, 1$; Eintrag $1, 1$ (Zeile 1, Spalte 1) in Matrix S^1 ist 1; Eintrag $3, 1$ in Matrix S^2 ist 3; die Einträge werden dual dargestellt, $x_1x_2 = 01$ und $x_3x_4 = 11$ und zusammengefügt: $x_1x_2x_3x_4 = 0111$. Zuletzt wird dieser 4-Block noch permutiert, $P_f(0111) = 1110$, und das ist das Ergebnis

$$f(0100, 0010\,0000) = 1110.$$

b) Analog wie in a) erhalten wir aus 1110 nach der Expansionspermutation

$$y_1 \dots y_8 = 0111\,1101 \oplus 0001\,0010 = 0110\,1111.$$

Es folgt die Auswahl von Eintrag an Position $1, 4$ in Matrix S^1 $(= 2)$ und Eintrag an Position $4, 4$ in Matrix S^2 $(= 3)$; die Einträge werden dual dargestellt, $x_1x_2 = 10$ und $x_3x_4 = 11$, zusammengefügt, permutiert und am Ende ist

$$f(1110, 0001\,0010) = 0111.$$

■

Die beiden Matrizen S^1 und S^2 stellen nichts anderes als Wertetabellen für die zugehörigen nichtlinearen Funktionen dar. Die acht Eingangsbits $x_1 \dots x_8$ (nach der E-Box und der Schlüsseladdition) werden dabei auf vier Ausgangsbits $y_1y_2y_3y_4$ (vor der P-Box) wie folgt abgebildet:

$$\begin{aligned}
y_1 &= x_2 + x_1x_2 + x_1x_3 + x_4 + x_1x_2x_3x_4 \\
y_2 &= 1 + x_1 + x_1x_2 + x_3 + x_1x_3 + x_1x_4 + x_1x_2x_4 + x_1x_2x_3x_4 \\
y_3 &= x_1 + x_2 + x_1x_3 + x_1x_2x_3 + x_4 + x_1x_4 + x_3x_4 + x_1x_3x_4 + x_1x_2x_3x_4 \\
y_4 &= x_1 + x_3 + x_1x_4 + x_2x_4 + x_1x_2x_4 + x_3x_4 + x_1x_3x_4
\end{aligned}$$

wobei Addition und Multiplikation in $\mathbb{Z}_2$ auszuführen sind. Die Implementierung mithilfe der Wertetabellen ist effektiver als das Auswerten dieser Formeln.

Wie kommt man auf diese Formeln? — Sei x eine Dezimalzahl zwischen 0 und 15 mit Dualdarstellung $x = (x_1x_2x_3x_4)_2$ und eine beliebige Funktion $F : \mathbb{Z}_2^4 \to \mathbb{Z}_2$ sei durch ihre Werte $F(x)$ für $x \in \mathbb{Z}_2^4$ gegeben. Dann können wir, analog wie bei der disjunktiven Normalform, Minterme $m_y : \mathbb{Z}_2^4 \to \mathbb{Z}_2$ definieren, die $m_y(y) = 1$ und $m_y(x) = 0$, $x \neq y$ erfüllen:

$$\begin{aligned} m_0(x) &= (x_1+1)(x_2+1)(x_3+1)(x_4+1), \\ m_1(x) &= x_1(x_2+1)(x_3+1)(x_4+1), \\ &\vdots \\ m_{15}(x) &= x_1x_2x_3x_4. \end{aligned}$$

Dann gilt

$$F(x) = \sum_{y=0}^{15} F(y)m_y(x) = F(0)(x_1+1)(x_2+1)(x_3+1)(x_4+1) + \cdots + F(15)x_1x_2x_3x_4.$$

(Achtung: Das Plus steht hier für die Addition modulo zwei und nicht, wie in der Schaltalgebra, für die logische ODER-Verknüpfung.)

Beispiel 4.3 (→CAS) S-DES
Verschlüsseln Sie den Klartext $1 = (0000\,0001)_2$ mit S-DES und dem Schlüssel $(34)_{10} = (0000100010)_2$.

Lösung zu 4.3 Die beiden notwendigen Teilschlüssel $k^1 = 0010\,0000$ und $k^2 = 0001\,0010$ wurden schon in Beispiel 4.1 bestimmt. Die Verschlüsselungsvorschrift lautet

$$(IP^{-1} \circ F_{k^2} \circ \Theta \circ F_{k^1} \circ IP)(0000\,0001),$$

d.h. es sind folgende Operationen durchzuführen: (a) auf 0000 0001 wird zunächst die Eingangspermutation IP angewandt; (b) das Ergebnis wird in die erste Feistel-Box F_{k^1} eingegeben; (c) danach vertauscht Θ linke und rechte Hälfte des Blocks, der aus der ersten Feistel-Box kommt; (d) das Ergebnis wird in die zweite Feistel-Box F_{k^2} eingegeben; (e) der Block, der aus der zweiten Feistel-Box kommt, wird noch der Ausgangspermutation IP^{-1} unterworfen – fertig ist der Geheimtext. Also (a) Eingangspermutation:

$$IP(0000\,0001) = 0000\,0100,$$

das kommt ...
(b) ... in die erste Feistel-Box: Die Eingangsbits werden in linke und rechte Hälfte aufgeteilt: $l = 0000$, $r = 0100$. Mithilfe Beispiel 4.2 a) folgt

$$F_{k_1}(0000, 0100) = (0000 \oplus f(0100, 0010\,0000), 0100) = (1110, 0100).$$

(c) Vertauschen von Links und Rechts:

$$\Theta(1110, 0100) = 0100\,1110,$$

weiter ...
(d) ...in die zweite Feistel-Box: Aufteilen in linke und rechte Hälfte: $l = 0100$, $r = 1110$, mithilfe Beispiel 4.2 b) erhalten wir

$$\begin{aligned} F_{k_2}(0100, 1110) &= (0100 \oplus f(1110, 00010010), 1110) = (0100 \oplus 0111, 1110) \\ &= (0011, 1110). \end{aligned}$$

(e) Ausgangspermutation:

$$IP^{-1}(0011\,1110) = 1011\,1001.$$

Das Ergebnis ist dezimal gleich 185. ■

4.2 Kryptoanalyse moderner Blockchiffren

Wir wollen uns in diesem Abschnitt etwas näher mit der Kryptoanalyse moderner Blockchiffren beschäftigen und die zwei wichtigsten Verfahren, die differentielle und die lineare Kryptoanalyse, besprechen. Ziel ist es einen kleinen Einblick in dieses Gebiet zu bekommen und auch die kryptographische Bedeutung der einzelnen Bauteile eines modernen Algorithmus besser zu verstehen.

4.2.1 Statistische Eigenschaften der S-Box

Um die Qualität eines Verschlüsselungsverfahrens (bzw. seiner Bestandteile) besser beurteilen zu können, führt man die sogenannte **Abhängigkeitsmatrix** A einer Funktion $F : \mathbb{Z}_2^n \to \mathbb{Z}_2^m$ ein. Sie ist eine $n \times m$ Matrix und der Eintrag A_{ij} gibt die Wahrscheinlichkeit an, dass sich bei einer Änderung des $i$ten Eingangsbits das $j$te Ausgangsbit ändert (unter der Annahme, dass die Eingangsbits gleichverteilt sind). D.h., A_{ij} ist die Anzahl der Fälle, in denen sich $y_j = F_j(x)$ ändert, wenn x_i zwischen 0 und 1 wechselt, und alle anderen Eingangsbits durchlaufen werden, geteilt durch 2^{n-1}:

$$A_{ij} = \frac{1}{2^{n-1}} \sum_{x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n \in \{0,1\}} (1 - \delta_{F_j(x_1,\ldots,x_{i-1},0,x_{i+1},\ldots,x_n),F_j(x_1,\ldots,x_{i-1},1,x_{i+1},\ldots,x_n)})$$

Hierbei ist $F(x) = (F_1(x), F_2(x), \ldots, F_m(x))$ und $\delta_{0,0} = \delta_{1,1} = 1$ bzw. $\delta_{0,1} = \delta_{1,0} = 0$ ist das Kronecker-δ.

Für eine große Blockgröße n kann A nicht mehr exakt berechnet werden, man kann aber versuchen, mit gleichverteilten Zufallsbits die Wahrscheinlichkeiten näherungsweise zu bestimmen.

Folgende Eigenschaften sind offensichtlich:

- $A_{ij} = 0$ falls das $j$te Ausgangsbit nicht vom $i$ten Eingangsbit abhängt.

- $A_{ij} = 1$ falls das $j$te Ausgangsbit sich bei jeder Änderung des $i$ten Eingangsbits ändert.

Somit können wir die am Anfang dieses Studienbriefs erwähnten Eigenschaften einer Verschlüsselungsvorschrift sofort von A ablesen:

Satz 4.4 Eine Funktion F ist vollständig genau dann, wenn $A_{ij} > 0$ für alle i, j gilt. Weiters erfüllt F die Lawineneigenschaft, genau dann wenn $A_{ij} \approx \frac{1}{2}$ für alle i, j gilt.

Wenn die Verschlüsselungsvorschrift von einem Schlüssel k abhängt, $F = E_k$, so erhält man im Allgemeinen für jeden Schlüssel eine andere Abhängigkeitsmatrix. Diese Eigenschaften müssen dann unabhängig vom Schlüssel gelten.

Beispiel 4.5 Abhängigkeitsmatrix
Mit $F(x) = (1, x_1, x_2 + x_3, x_1x_2 + x_3x_4)$ erhalten wir

$$\begin{pmatrix} 0 & 1 & 0 & 0.5 \\ 0 & 0 & 1 & 0.5 \\ 0 & 0 & 1 & 0.5 \\ 0 & 0 & 0 & 0.5 \end{pmatrix}.$$

Die erste Spalte ist klar, denn $F_1(x) = 1$ hängt von keiner der 4 Variablen ab. Ebenso ist die zweite Spalte leicht einzusehen, denn nur eine Änderung von x_1 ändert $F_2(x) = x_1$. Die dritte Spalte folgt, da $F_3(x) = x_2 + x_3$ sich genau dann ändert, wenn sich x_2 oder x_3 ändern. Die letzte Spalte folgt, da sich $F_4(x) = x_1x_2 + x_3x_4$ bei Änderung von x_1 genau dann ändert, wenn $x_2 = 1$ und damit in 50% der Fälle. Analog für x_2, x_3, x_4.

Dass im Fall einer affinen Komponente F_j nur 0 und 1 in der $j$ten Spalte stehen ist kein Zufall.

Im Fall $A_{ij} = 0$ gilt $F_j(x) = F_j(\hat{x}^i)$ mit $\hat{x}^i = (x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$. Im Fall $A_{ij} = 1$ gilt $F_j(x) = F_j(\hat{x}^i) + x_i$. Besteht eine Zeile von $(A_{1j}, \ldots, A_{nj})$ nur aus den Werten 0 und 1, so können wir daraus F_j vollständig rekonstruieren: $F_j(x) = F_j(0) + \sum_{i=1}^{n} A_{ij}x_i$. Insbesondere erhalten wir:

Satz 4.6 Es gilt $A_{1j}, A_{2j}, \ldots, A_{nj} \in \{0, 1\}$ genau dann, wenn die $j$te Komponente F_j affin in den Eingangsbits ist. Insbesondere ist F genau dann affin, wenn $A_{ij} \in \{0, 1\}$ für alle i, j ist. In diesem Fall gilt

$$F(x) = F(0) + A^T x.$$

Steht in jeder Zeile von A genau eine 1 (und sonst überall 0), so ist F eine Permutation.

Ist x^0 ein konstanter Vektor, so haben $F(x + x^0)$ und $F(x) + x^0$ die gleiche Abhängigkeitsmatrix wie F.

Beispiel 4.7 Abhängigkeitsmatrix
Für die Abhängigkeitsmatrix der S-Box des S-DES erhält man

$$\begin{pmatrix} 0.62 & 0.38 & 0 & 0 \\ 0.62 & 0.38 & 0 & 0 \\ 0.38 & 0.62 & 0 & 0 \\ 0.88 & 0.38 & 0 & 0 \\ 0 & 0 & 0.38 & 0.75 \\ 0 & 0 & 0.88 & 0.25 \\ 0 & 0 & 0.62 & 0.75 \\ 0 & 0 & 0.38 & 0.75 \end{pmatrix}$$

und man erkennt sehr schön, dass sie aus zwei Blöcken S_1 und S_2 aufgebaut ist. Für $f(x, k)$ erhält man (unabhängig vom Schlüssel):

$$\begin{pmatrix} 0.38 & 0.75 & 0.38 & 0.62 \\ 0.62 & 0.75 & 0.38 & 0.38 \\ 0.38 & 0.25 & 0.88 & 0.88 \\ 0.38 & 0.75 & 0.62 & 0.62 \end{pmatrix}$$

Sie ist also vollständig. Für den gesamten S-DES ergibt sich (immer noch unabhängig vom Schlüssel):

$$\begin{pmatrix} 0.62 & 0.38 & 0.62 & 0 & 0 & 0.75 & 1 & 0 \\ 0.62 & 0.38 & 0.38 & 1 & 0 & 0.75 & 0 & 0 \\ 0.88 & 0.38 & 0.88 & 0 & 1 & 0.25 & 0 & 0 \\ 0.55 & 0.42 & 0.42 & 0.38 & 0.38 & 0.38 & 0.62 & 0.75 \\ 0.41 & 0.41 & 0.62 & 0.38 & 0.88 & 0.59 & 0.88 & 0.25 \\ 0.38 & 0.62 & 0.38 & 0 & 0 & 0.75 & 0 & 1 \\ 0.59 & 0.59 & 0.38 & 0.38 & 0.62 & 0.56 & 0.62 & 0.75 \\ 0.50 & 0.59 & 0.53 & 0.62 & 0.38 & 0.62 & 0.38 & 0.75 \end{pmatrix}$$

Das schaut nicht so gut aus (weder das Vollständigkeits- noch das Lawinenkriterium sind erfüllt) und liegt daran, dass der S-DES nur zwei Runden hat! Durch Hinzufügen einer weiteren Runde verbessert sich das Bild deutlich (mit Schlüssel $k = 0$):

$$\begin{pmatrix} 0.55 & 0.53 & 0.47 & 0.59 & 0.38 & 0.47 & 0.59 & 0.56 \\ 0.48 & 0.42 & 0.47 & 0.42 & 0.42 & 0.45 & 0.55 & 0.38 \\ 0.48 & 0.62 & 0.47 & 0.41 & 0.62 & 0.56 & 0.41 & 0.59 \\ 0.52 & 0.48 & 0.45 & 0.42 & 0.47 & 0.47 & 0.48 & 0.45 \\ 0.53 & 0.50 & 0.53 & 0.62 & 0.47 & 0.50 & 0.48 & 0.56 \\ 0.53 & 0.48 & 0.52 & 0.59 & 0.53 & 0.55 & 0.50 & 0.62 \\ 0.42 & 0.50 & 0.50 & 0.53 & 0.47 & 0.47 & 0.55 & 0.47 \\ 0.52 & 0.53 & 0.42 & 0.48 & 0.52 & 0.48 & 0.53 & 0.55 \end{pmatrix}$$

Mit anderen Schlüsseln erhält man leicht unterschiedliche Ergebnisse. Wir sehen also anschaulich, wie die Anzahl der Runden die Sicherheit erhöht. Das ist in Abbildung 4.3 auch als Bild veranschaulicht, bei zunehmender Anzahl der Runden liegen die Werte immer näher bei einem mittleren Grauwert.

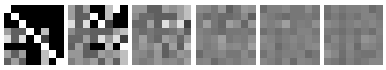

Abbildung 4.3: Die Abhängigkeitsmatrix beim S-DES für verscheidenen Rundenanzahl $(1, 2, 3, 4, 5)$ als Bild dargestellt (der Grauwert der einzelnen Pixel entspricht dem Wert der Abhängigkeitsmatrix; Schwarz=0 und Weiss=1).

Achtung, die Abhängigkeitsmatrix alleine ist kein ausreichendes Beurteilungskriterium. Sie erkennt z.B. nicht, wenn alle Ausgangsbits gleich sind.

4.2.2 Naive Kryptoanalyse der f-Box

Versuchen wir uns nun an der Kryptoanalyse des S-DES. Wir analysieren dazu als erstes die f-Box. Angenommen wir kennen die Eingangswerte x und die Ausgangswerte y und wollen daraus den Teilschlüssel k ermitteln. Dazu können wir die Eingangswerte bis vor die Schlüsseladdition und die Ausgangswerte bis vor die S-Box verfolgen. Nun müssen wir die Ausgangswerte durch die S-Box zurückverfolgen. Da die S-Box nicht invertierbar ist, bekommen wir hier mehrere Werte. Dann wissen wir die möglichen Werte vor der Schlüsseladdition und können daraus die möglichen Schlüssel berechnen.

Beispiel 4.8 (→CAS) Kryptoanalyse der f-Box des S-DES
Bestimmen Sie aus

$$f(0100, k) = 1110$$

die möglichen geheimen Teilschlüssel k der f-Box des S-DES.

Lösung zu 4.8 Es ist also $x = x_1x_2x_3x_3 = 0100$ und $y = y_1y_2y_3y_4 = 1110$. Damit sind die Werte vor der Schlüsseladdition gleich

$$x_4x_1x_2x_3\, x_2x_3x_4x_1 = 0010\,1000$$

und die Werte am Ausgang der S-Box sind

$$y_4y_1y_3y_2 = 0111.$$

Aus der Wertetabelle für die beiden einzelnen S-Boxen

x	$S^1(x)$	$S^2(x)$
0000	**01**	00
0001	11	10
0010	00	01
0011	10	00
0100	11	10
0101	**01**	01
0110	10	**11**
0111	00	**11**
1000	00	**11**
1001	11	10
1010	10	00
1011	**01**	01
1100	**01**	01
1101	11	00
1110	11	00
1111	10	**11**

lesen wir die möglichen Eingangswerte ab:

0000 0110	0000 0111	0000 1000	0000 1111
0101 0110	0101 0111	0101 1000	0101 1111
1011 0110	1011 0111	1011 1000	1011 1111
1100 0110	1100 0111	1100 1000	1100 1111

Differenz mit dem bekannten Wert 0010 1000 vor der Schlüsseladdition gibt uns 16 mögliche Schlüssel:

0010 1110	0010 1111	0010 0000	0010 0111
0111 1110	0111 1111	0111 0000	0111 0111
1001 1110	1001 1111	1001 0000	1001 0111
1110 1110	1110 1111	1110 0000	1110 0111

■

Da die beiden S-Boxen 4 Bits auf 2 Bits abbilden, haben wir im Mittel von jeder S-Box 4 mögliche Urbilder und somit, wie im Beispiel, 16 mögliche Teilschlüssel. Mit weiteren Paaren x, y können wir die Anzahl der Möglichkeiten weiter einschränken und so den Schlüssel erhalten. Bei zwei Runden funktioniert das Verfahren unverändert, da wir die Eingangs- und Ausgangswerte der f-Box in der ersten Runde kennen! In der Tat können wir IP, IP^{-1} leicht herausrechnen und somit ignorieren. Dann gilt:

Runde	links	rechts
0	l	r
1	$l \oplus f_1$	r
2	$r \oplus f_2$	$l \oplus f_1$

wobei f_j das Ergebnis der f-Box in der j'ten Runde bezeichnet. Also ist die Eingabe der ersten f-Box die rechte Seite der Eingabe und die Ausgabe der ersten f-Box die mit l XOR verknüpfte rechte Seite der Ausgabe. Mit diesen Informationen können wir, wie im Beispiel demonstriert, die erste f-Box angreifen und den ersten Teilschlüssel erhalten. Die fehlenden 2 Bits des Schlüssel können dann entweder mit Brute-Force bestimmt werden oder indem man auch die zweite f-Box angreift.

Beispiel 4.9 (→CAS) Kryptoanalyse des S-DES
Bestimmen Sie alle möglichen Schlüssel k mit

$$SDES_k(229) = 38.$$

Lösung zu 4.9 Es gilt $x = 1110\,0101$ und nach der Eingangspermutation $IP(x) = 1111\,0100$. Analog $y = 0010\,0110$ und $IP(y) = 0110\,0001$. Damit ist $l = 1111$, $r = 0100$ und $r \oplus f_2 = 0110$, $l \oplus f_1 = 0001$ bzw. $f_1 = 1110$, $f_2 = 0010$. Der Eingangswert für die zweite f-Box ist also $l \oplus f_1 = 0001$ und somit insgesamt

$$f(0100, k^1) = 1110, \qquad f(0001, k^2) = 0010$$

mit den Teilschlüsseln $k^1 = k_1k_7k_9k_4\,k_8k_3k_{10}k_6$ und $k^2 = k_8k_3k_6k_5\,k_{10}k_2k_9k_1$. Die erste f-Box haben wir bereits im letzten Beispiel analysiert und 16 Möglichkeiten für k^1 bekommen. Zusammen mit den zwei fehlenden Schlüsselbits k_2 und k_5 gibt das 64 mögliche Schlüssel die wir durchprobieren können.

Wir können alternativ auch die zweite f-Box analysieren. Wie zuvor erhalten wir den Wert vor der Schlüsseladdition als $1000\,0010$ und am Ausgang der S-Box als $00\,10$. Damit ergeben sich anhand der Tabelle aus dem letzten Beispiel die drei Möglichkeiten 0010, 0111, 1000 für die erste Hälfte nach der Schlüsseladdition und die drei Möglichkeiten 0001, 0100, 1001 für die zweite Hälfte. Damit sind 1010, 1111, 0000 die Möglichkeiten für die erste Hälfte von k^2 und 0011, 0110, 1011 die Möglichkeiten für die zweite Hälfte:

10100011	10100110	10101011
11110011	11110110	11111011
00000011	00000110	00001011

Da k^1 und k^2 vom gleichen Schlüssel abstammen, muss $k^1_1 = k^1_8$, $k^1_6 = k^2_2$, $k^1_8 = k^2_3$, $k^1_5 = k^2_1$, $k^1_3 = k^2_7$ und $k^1_7 = k^2_5$ gelten. Somit verbleiben vier Möglichkeiten für k:

$$258, 330, 703, 522.$$

Das sind alle möglichen Schlüssel. ■

Damit ist der S-DES vollständig gebrochen. Das Gleiche gilt für jeden anderen Algorithmus, der aus einem Feistel-Netzwerk aus maximal 2 Runden besteht. Der Aufwand für den Angriff entspricht dabei der Eingangsgröße der S-Boxen.

Bei mehreren Runden scheitert das Verfahren jedoch, da mit jedem Durchlauf der S-Box die Anzahl der Urbilder, die zurückverfolgt werden müssen, exponentiell ansteigt. Wird beim S-DES die f-Box zweimal durchlaufen, so sind es im Mittel schon $16^2 = 256$ Urbilder.

4.2.3 Differentielle Kryptoanalyse

Ein Verfahren, dass sich auch auf mehrere Runden erweitern läßt, ist die **differentiellen Kryptoanalyse**. Wir betrachten zunächst einfachheitshalber wieder nur eine Runde.

Die Idee ist ähnlich wie zuvor. Wir versuchen die S-Box in die Zange zu nehmen um ihr den Teilschlüssel entlocken zu können. Dazu verfolgen wir die Eingangswerte wieder bis vor die Schlüsseladdition und von der anderen Seite die Ausgangswerte bis zum Ausgang der S-Box. Nun blockiert die Schlüsseladdition (da wir den Schlüssel ja nicht kennen) unseren Zugriff auf die S-Box (siehe Abbildung 4.4). Haben wir zwei Paare zur Verfügung, so können wir zu Differenzen[1] wechseln. Diese werden von den linearen Komponenten (E-Box und P-Box) nicht verändert und bei der Schlüsseladdition kürzt sich der Schlüssel heraus! Somit kennen wir die Differenzen vor und nach der S-Box und unsere Zange kann zupacken. Nun suchen

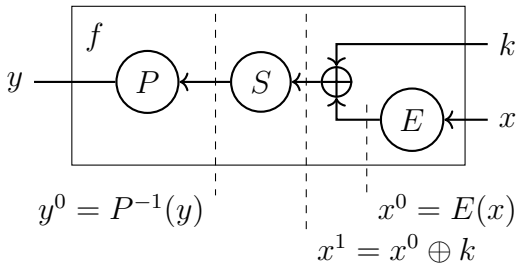

Abbildung 4.4: Idee der differentiellen Kryptoanalyse: x^0 vor der Schlüsseladdition und y^0 am Ausgang der S-Box können ohne Kenntnis des Schlüssels k aus Eingang x und Ausgang y berechnet werden. x^1 benötigt den Schlüssel. Kennt man zwei Paare (x, y) und $(\tilde{x}, \tilde{y})$, so gilt $x^1 \oplus \tilde{x}^1 = x^0 \oplus \tilde{x}^0$ und man kennt die Differenz vor und nach der S-Box.

wir Paare die die passende Ausgangsdifferenz erzeugen und berechnen damit die möglichen Schlüssel.

Wieder ist es entscheidend, dass das Auffinden der Paare auf die kleinen S-Boxen S^j heruntergebrochen werden kann.

[1] Wir rechen in $\mathbb{Z}_2$: Wegen $-x = x$ fallen Addition und Subtraktion zusammen.

Beispiel 4.10 (→CAS) Differentielle Kryptoanalyse der f-Box des S-DES
Bestimmen Sie aus

$$f(0100, k) = 1110, \qquad f(0111, k) = 1001$$

den geheimen Teilschlüssel k der f-Box des S-DES.

Lösung zu 4.10 Wir haben also die beiden Paare $x = 0100$, $y = 1110$ und $\tilde{x} = 0111$, $\tilde{y} = 1001$. Mit der Expansionspermutation wird daraus

$$x^0 = 0010\,1000, \qquad \tilde{x}^0 = 1011\,1110,$$

und das ist der Stand vor der Schlüsseladdition. Die Differenz ist

$$x^0 \oplus \tilde{x}^0 = 1001\,0110$$

und das stimmt auch noch, bevor es in die S-Box geht:

$$(x^0 \oplus k) \oplus (\tilde{x}^0 \oplus k) = x^0 \oplus k \oplus \tilde{x}^0 \oplus k = x^0 \oplus \tilde{x}^0.$$

Machen wir bei den Ausgangsbits die Permutation P_f rückgängig, so muss das Ergebnis der S-Box gleich

$$y^0 = 0111, \qquad \tilde{y}^0 = 1100, \qquad y^0 \oplus \tilde{y}^0 = 1011$$

sein. Nun geht es darum, Eingänge für die S-Box zu finden, die sich um $1001\,0110$ unterscheiden und deren Ausgangswerte sich um 1011 unterscheiden.

Da die S-Box aus zwei unabhängigen Teilen S^1 und S^2 besteht, können wir unser Problem in zwei Teile zerlegen. Beginnen wir mit der ersten 4-Bit-Hälfte. Wir lassen x also alle möglichen 2^4 Werte durchlaufen, berechnen $x' = x \oplus 1001$, und transformieren beide Werte gemäß S^1 (Permutation und Substitution):

x	x'	$S^1(x)$	$S^1(x')$	$S^1(x) \oplus S^1(x')$
0000	1001	01	11	**10**
0001	1000	11	00	11
0010	1011	00	01	01
0011	1010	10	10	00
0100	1101	11	11	00
0101	1100	01	01	00
0110	1111	10	10	00
0111	1110	00	11	11
1000	0001	00	11	11
1001	0000	11	01	**10**
1010	0011	10	10	00
1011	0010	01	00	01
1100	0101	01	01	00
1101	0100	11	11	00
1110	0111	11	00	11
1111	0110	10	10	00

Wir erhalten also zwei Treffer: $x = 0000$ und $x = 1001$. XOR-Verknüpfung mit den ersten vier Bits von x^0 gibt zwei Möglichkeiten für die ersten vier Schlüsselbits: 0010 und 1011. Indem wir beide probieren, erhalten wir 0010 als die gesuchten ersten 4 Bits.

Analog erhalten wir mit S^2 sechs Möglichkeiten für die letzten vier Schlüsselbits: 1000, 1010, 1100, 1110, 0000 und 0110. Indem wir diese probieren, finden wir 0000 und 0110 als mögliche letzten 4 Bits. Insgesamt ist der geheimen Schlüssel also $k = 0010\,0000$ oder $k = 0010\,1110$.

Hat man mehrere Paare zur Verfügung, so kann man sich anhand folgender **Differenzentabelle** Paare aussuchen, die zu möglich wenigen Werten führen.

	S^1				S^2			
Δx	00	01	10	11	00	01	10	11
0000	16	0	0	0	16	0	0	0
0001	0	2	10	4	2	8	2	4
0010	0	10	6	0	0	6	4	6
0011	2	4	0	10	4	2	8	2
0100	2	4	8	2	2	0	10	4
0101	10	0	4	2	2	4	2	8
0110	0	2	2	12	0	10	0	6
0111	4	10	2	0	8	2	4	2
1000	2	4	8	2	4	6	0	6
1001	8	2	2	4	8	2	4	2
1010	4	2	2	8	2	0	10	4
1011	2	8	4	2	0	6	4	6
1100	8	2	2	4	0	6	4	6
1101	2	4	8	2	6	0	6	4
1110	2	8	4	2	10	4	2	0
1111	4	2	2	8	2	8	2	4

Hier stehen für jede mögliche Differenz der x Werte (Zeile) und jede mögliche Differenz der y Werte (Spalte) wie oft sie auftreten. In unserem Fall waren die Differenzen 1001,10 für S^1, was 2 Möglichkeiten laut der letzten Tabelle gibt und 0110,11 für S^2, was 6 Möglichkeiten gibt. Gut sind also Paare bei denen diese Werte möglichst klein sind. Können wir uns den Klartext aussuchen, so können wir versuchen x aus einer Spalte mit kleinen Werten vorzugeben und dann auf einen guten y Wert hoffen. ■

Wie schon zuvor erwähnt, funktioniert das Verfahren bei zwei Runden unverändert. Bei mehreren Runden muss der Angriff verfeinert werden, und man benötigt passend gewählte Klartextpaare, damit der Angriff durchführbar wird. Die Idee ist es, die Differenzen durch die Runden zu verfolgen. Man beginnt also mit einer Eingangsdifferenz Δ_0 und betrachtet die (möglichen) Differenzen Δ_j nach j Runden. Diese Differenzen sind wie zuvor unabhängig von den Teilschlüsseln. Eine solche Folge von Differenzen wird als **Charakteristik** bezeichnet. Wenn man nun Δ_0 vorgibt, so kann man sich natürlich nicht erwarten, dass ein passendes Startpaar $(x, \tilde{x})$ mit $x \oplus \tilde{x} = \Delta_0$ auch zu dieser Charakteristik führt. In der Tat hängt die Folge der Differenzen sowohl von x und $\tilde{x}$ selbst, als auch vom Schlüssel ab. Sie ist bei einem Angriff also nicht vorhersagbar. Wir können aber anhand der Analyse der Differenzentabellen der S-Boxen (so wie am Ende des letzten Beispiels) versuchen, Charakeristiken zu finden, die mit vergleichsweise hohen Wahrscheinlichkeiten auftreten.

Beispiel 4.11 Bei jedem Feistel-Netzwerk (ohne IP, IP^{-1}) gibt es immer die Einrundencharachteristik $\Delta_0 = (l, 0)$, $\Delta_1 = (0, l)$ mit Wahrscheinlichkeit 1.

Hat man so eine Charakeristik $\Delta_0, \Delta_1, \ldots$, identifiziert, läuft der Angriff wie folgt ab: Man erzeugt Klartextpaare mit der passenden Eingangsdifferenz Δ_0. Unter der Annahme, dass das gewählte Paar die gewählte Charakteristik realisiert, kann man wie in Beispiel 4.10 die letzte f-Box angreifen und mögliche Werte für den letzten Teilschlüssel bestimmen. Macht man das mit mehreren Paaren, so sollten die erwarteten Differenzen vergleichsweise häufig auftreten und damit auch der gesuchte Teilschlüssel.

Um diesen Angriff auf den S-DES mit 4 (statt 2) Runden anzuwenden verschaffen wir uns zuerst einen Überblick über die grobe Struktur. Gegeben sei ein Klar-/Geheimtextpaar $y = SDES_k(x)$. Wir bezeichnen mit $(l, r) = IP(x)$ die linke/rechte Hälfte nach der Eingagnspermutation und mit $(l_y, r_y) = IP(y)$ die linke/rechte Hälfte vor der Ausgangspermutation. Dann gilt:

	links	rechts
Eingang	l	r
Runde 1	$l \oplus f_1$	r
Runde 2	$r \oplus f_2$	$l \oplus f_1$
Runde 3	$l \oplus f_1 \oplus f_3$	$r \oplus f_2$
Runde 4	$r \oplus f_2 \oplus f_4$	$l \oplus f_1 \oplus f_3$
Ausgang	l_y	r_y

wobei f_j das Ergebnis der f-Box in der j'ten Runde bezeichnet. Die gleiche Tabelle gilt natürlich auch für die zugehörigen Differenzen. Unser Plan ist die f-Box in der letzten Runde anzugreifen. Die Eingabedifferenz $\Delta l \oplus \Delta f_1 \oplus \Delta f_3 = \Delta r_y$ ist bekannt, aber die Ausgabedifferenz $\Delta f_4 = \Delta r \oplus \Delta f_2 \oplus \Delta l_y$ ist unbekannt, da wir Δf_2 nicht kennen. Wir benötigen für einen Angriff also Δf_2.

Beispiel 4.12 Finden Sie eine passende Charakteristik zur differentiellen Kryptoanalyse des S-DES mit 4 (statt 2) Runden.

Lösung zu 4.12 Beginnen wir z.B. mit $\Delta x = 0100\,0100$. Dann gilt nach der Eingangspermutation $\Delta l = 1100$ und $\Delta r = 0$. Wir starten also mit der Einrundencharakteristik, so dass $\Delta f_1 = 0$ gilt und wir sofort in die nächste Runde wechseln können.

Der Eingang der zweiten f-Box ist $\Delta l \oplus \Delta f_1 = \Delta l$. Nach der E-Box gilt $E(\Delta l) = 0110\,1001$ und da diese Differenz durch die Schlüsseladdition ja nicht verändert wird, ist das der Eingang für die zweite S-Box. Unsere Wahl wurde so getroffen, dass sowohl die linke Hälfte 0110 (die S^1 durchläuft) als auch die rechte Hälfte 1001 (die S^2 durchläuft) mit hoher Wahrscheinlichkeit zu bestimmten Ausgangsdifferenzen führt. Ein Blick auf die Differenzentabelle aus dem letzen Beispiel zeigt, dass die Ausgangsdifferenz von S^1 mit Wahrscheinlichkeit $\frac{12}{16} = \frac{3}{4}$

gleich 11 und die Ausgangsdifferenz von S^2 mit Wahrscheinlichkeit $\frac{8}{16} = \frac{1}{2}$ gleich 00 ist. Nach der P-Box erwarten wir also die Ausgangsdifferenz $\Delta f_2 = 1001$ mit Wahrscheinlichkeit $\approx \frac{3}{4} \cdot \frac{1}{2} = \frac{3}{8}$, wenn wir davon ausgehen, dass beide Ereignisse annähernd unabhängig sind. ■

Damit kann der Angriff wie oben beschrieben durchgeführt werden: Man erzeugt Paare $(x, \tilde{x})$ mit der Differenz $x \oplus \tilde{x} = 0100\,0100$ und bittet sein Opfer diese Paare zu verschlüsseln um $(y, \tilde{y})$ zu erhalten. Nun bestimmen wir wie in Beispiel 4.10 mithilfe von

$$f(r_y, k^4) \oplus f(\tilde{r}_y, k^4) = \Delta r \oplus \Delta f_2 \oplus \Delta l_y = 1001 \oplus \Delta l_y$$

mögliche Werte für den letzten Teilschlüssel k^4 (die Probe muss entfallen, da wir nur Δf_2 (mit einer bestimmten Wahrscheinlichkeit) kennen, nicht aber f_2 und $\tilde{f}_2$). Der Teilschlüssel der bei mehreren Paaren am häufigsten auftritt sollte der gesuchte Teilschlüssel sein.

Bei einem Experiment wurde mit 10 zufällig gewählten Paaren der geheime Teilschlüssel in 32% aller Fälle gefunden. Mit 20 Paaren war dass in 51% der Fall und mit 50 Paaren in 78%.

Eine weitere Ausführung an einem anderen Beispiel ist in [Hay02] zu finden. Für weitere Details im Fall des echten DES sei auf [BS93] verwiesen.

4.2.4 Lineare Kryptoanalyse

Eine alternative Methode ist die **lineare Kryptoanalyse** [Mat93]. Die Idee ist, dass es bei einem affinen Verschlüsselungsalgorithmus affine Beziehungen der Form

$$x_{i_1} \oplus \cdots \oplus x_{i_{n_x}} \oplus y_{i_1} \oplus \cdots \oplus y_{i_{n_y}} \oplus c = k_{i_1} \oplus \cdots \oplus k_{i_{n_y}}$$

zwischen dem Klartext x, dem Geheimtext y und dem Schlüssel k gibt. Die Konstante $c \in \{0, 1\}$ wird dabei meist weggelassen, was bedeutet, dass die Gleichung dann entweder immer richtig ($c = 0$) oder immer falsch ($c = 1$) ist. Bei einem idealen Verschlüsselungsalgorithmus darf es eine solche Beziehung nicht geben, denn der Geheimtext sollte wie ein Zufallstext sein, was bedeutet, dass eine beliebige lineare Beziehung in durchschnittlich 50% der Fälle richtig, und in 50% der Fälle falsch sein wird. Hat man nun für einen Verschlüsselungsalgorithmus eine affine Beziehung gefunden, bei der die Wahrscheinlichkeit, dass sie gültig ist, größer $\frac{1}{2}$ ist, so wird folgender Angriff möglich: Man setzt eine (möglichst große) Anzahl aus Klar-/Geheimtextpaaren ein und erhält eine Beziehung zwischen den Schlüsselbits, die mit hoher Wahrscheinlichkeit richtig ist. Wie viele Paare man benötigt, um (statistisch gesehen) eine richtige Entscheidung zu treffen, hängt davon ab, wie weit die Wahrscheinlichkeit unserer Gleichung von $\frac{1}{2}$ entfernt ist. Diese Abweichung wird als **Bias** bezeichnet. Hat man genügend viele Beziehungen, so kann man die fehlenden Informationen mit Brute-Force erhalten.

Beispiel 4.13 Lineare Krypoanalyse: XOR
Die Verschlüsselung operiere auf zwei Bits durch bitweises XOR mit dem Schlüssel: $E_k(x) = x \oplus k$. Finden Sie passende bzw. unpassende lineare Beziehungen.

Lösung zu 4.13 Die einzigen beiden gültigen Beziehungen in diesem Fall sind

$$x_1 \oplus y_1 = k_1 \qquad \text{und} \qquad x_2 \oplus y_2 = k_2$$

und beide gelten mit Wahrscheinlichkeit 1. Wir brauchen also nur ein Klar-/Geheimtextpaar um den Schlüssel zu bestimmen. Alle anderen Beziehungen (bis auf die Summe der beiden obigen Gleichungen) wie z.B.

$$x_1 = k_1, \qquad x_1 \oplus y_2 = k_1, \qquad \text{etc.}$$

sind mit Wahrscheinlichkeit $\frac{1}{2}$ erfüllt und somit unbrauchbar um Informationen über den Schlüssel zu erhalten.

Das ist auch so gut zu verstehen: Addieren Sie auch nur einen einzigen Term (ein einzelnes x_j, y_j oder k_j) zu einer bestehenden Gleichung, so wirkt das wie ein Zufallsbit. Ist dieses Bit gleich 0, so ändert sich nichts und ist es gleich 1, so vertauschen sich richtig und falsch. Mit dem Ergebnis, dass die Wahrscheinlichkeit nun $\frac{1}{2}$ ist, egal was der Wert vorher war. ■

Beispiel 4.14 Lineare Krypoanalyse: AND
Die *Verschlüsselung* operiere auf zwei Bits durch bitweises AND mit dem Schlüssel: $E_k(x) = x \wedge k$. Finden Sie passende bzw. unpassende lineare Beziehungen.

Lösung zu 4.14 In diesem Fall ist die Operation nichtlinear (und auch nicht invertierbar, also eigentlich nicht zur Verschlüsselung geeignet). Versuchen wir wieder die Beziehungen aus dem letzten Beispiel

$$x_1 \oplus y_1 = k_1 \qquad \text{und} \qquad x_2 \oplus y_2 = k_2.$$

Beide gelten nun mit Wahrscheinlichkeit $\frac{1}{4}$. Addieren wir auf den linken Seiten 1,

$$x_1 \oplus y_1 \oplus 1 = k_1 \qquad \text{und} \qquad x_2 \oplus y_2 + 1 = k_2,$$

so gelten beide mit Wahrscheinlichkeit $\frac{3}{4}$. Beachten Sie, dass diese Wahrscheinlichkeit als Mittelwert über alle Schlüssel zu verstehen ist. Halten wir den Schlüssel fest, so erhalten wir für die erste Gleichung die Wahrscheinlichkeit $\frac{1}{2}$ im Fall $k_1 = 0$ und die Wahrscheinlichkeit 1 im Fall $k_1 = 1$. Ist also $k_1 = 1$, so liefert jedes Klar-/Geheimtextpaar den richtigen Schlüssel und im Fall $k_1 = 0$ erhalten wir ein Zufallsbit.

Die meisten anderen Relationen wie z.B.

$$x_1 = k_1, \qquad x_1 \oplus y_2 = k_1, \qquad \text{etc.}$$

sind mit Wahrscheinlichkeit $\frac{1}{2}$ erfüllt und somit unbrauchbar um Informationen über den Schlüssel zu erhalten. ■

Um diesen Angriff durchzuführen benötigt man also passende lineare Beziehungen. Das ist nicht so leicht wie es erscheint, denn eine zufällig gewählte lineare Beziehung ist meistens unbrauchbar. Der Weg solche zu finden geht deshalb, wie nicht schwer zu erraten, über die S-Boxen.

Beispiel 4.15 (→CAS) Lineare Kryptoanalyse: S-Box
Finden Sie mögliche lineare Beziehungen für die S-Box des S-DES.

Lösung zu 4.15 Wir berechnen uns zunächst eine **Biastabelle** für die beiden S-Boxen:

	S^1			S^2		
	y_1	y_2	$y_1 \oplus y_2$	y_1	y_2	$y_1 \oplus y_2$
0	−0.062	−0.062	0	0.062	0	0.062
x_1	0.062	0.062	0	−0.062	0	−0.062
x_2	0.062	0.062	0	0.062	0.12	−0.062
$x_1 \oplus x_2$	−0.062	−0.062	0	0.19	0.12	0.062
x_3	0.062	−0.31	0.12	−0.062	0.12	−0.19
$x_1 \oplus x_3$	−0.062	−0.19	−0.12	0.062	0.12	−0.062
$x_2 \oplus x_3$	−0.062	0.062	0.12	−0.31	0	0.19
$x_1 \oplus x_2 \oplus x_3$	0.062	−0.062	−0.12	0.062	0	0.062
x_4	0.062	0.062	0	0.062	0	0.062
$x_1 \oplus x_4$	−0.062	−0.062	0	−0.062	0	−0.062
$x_2 \oplus x_4$	0.19	0.19	0	0.062	−0.12	0.19
$x_1 \oplus x_2 \oplus x_4$	0.31	−0.19	0	0.19	−0.12	−0.19
$x_3 \oplus x_4$	0.19	0.062	−0.12	−0.062	−0.12	0.062
$x_1 \oplus x_3 \oplus x_4$	−0.19	−0.062	0.12	0.062	**0.38**	0.19
$x_2 \oplus x_3 \oplus x_4$	0.062	−0.062	−0.12	0.19	0	0.19
$x_1 \oplus x_2 \oplus x_3 \oplus x_4$	−0.062	0.062	**−0.38**	0.062	0	0.062

Zur besseren Übersicht haben wir sie als Matrix geschrieben, wobei die Zeile dem x-Anteil und die Spalte dem y-Anteil der Gleichung entspricht.

Für S^1 ist also die Beziehung mit dem (betragsmäßig) größten Bias

$$x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus y_1 \oplus y_2,$$

die in 12.5% aller Fälle 0 und in 87.5% aller Fälle 1 ergibt. Für S^2 ist die Beziehung mit dem größten Bias

$$x_1 \oplus x_3 \oplus x_4 \oplus y_2,$$

die in 12.5% aller Fälle 0 und in 87.5% aller Fälle 1 ergibt. ■

Beispiel 4.16 (→CAS) Lineare Kryptoanalyse des S-DES
Verwenden Sie die Beziehung

$$x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus y_1 \oplus y_2,$$

für den ersten Teil der S-Box $S^1 : x_1x_2x_3x_4 \mapsto y_1y_2$ beim S-DES um eine affine Beziehung für den S-DES zu finden.

Lösung zu 4.16 Es seien x die Eingagnsbits, y die Ausgangsbits und k die Schlüsselbits beim S-DES. Weiters seien X^1, Y^1 die Eingangs-/Ausgangswerte der ersten (von der rechten Hälfte) durchlaufenen S-Box und X^2, Y^2 die Eingangs-/Ausgangswerte der zweiten (von der linken Hälfte) durchlaufenen S-Box. Durch (rück-)verfolgen des Algorithmus erhalten wir

$$X^1 = E(R(IP(x))) \oplus k^1, \qquad Y^1 = P^{-1}(R(IP(y)) \oplus L(IP(x)))$$

und

$$X^2 = E(R(IP(y))) \oplus k^2, \qquad Y^2 = P^{-1}(L(IP(y)) \oplus R(IP(x))),$$

wobei E, P die E- bzw. P-Box der f-Box bezeichnet und L, R die linke bzw. rechte Hälfte. Explizit erhalten wir

$$X^1_1 = k_1 \oplus x_7, \quad X^1_2 = k_7 \oplus x_4, \quad X^1_3 = k_9 \oplus x_8, \quad X^1_4 = k_4 \oplus x_5$$

und

$$Y^1_1 = x_1 \oplus y_7, \quad Y^1_2 = x_2 \oplus y_4.$$

Durch Einsetzten in unsere Beziehung

$$X^1_1 \oplus X^1_2 \oplus X^1_3 \oplus X^1_4 \oplus Y^1_1 \oplus Y^1_2 = 1,$$

wird daraus

$$x_1 \oplus x_2 \oplus x_4 \oplus x_5 \oplus x_7 \oplus x_8 \oplus y_4 \oplus y_7 \oplus 1 = k_1 \oplus k_4 \oplus k_7 \oplus k_9.$$

Da die Gleichungen für X^j und Y^j immer richtig sind, ändern sich die Wahrscheinlichkeiten nicht und diese Beziehung ist ebenfalls in 87.5% aller Fälle richtig.

Indem wir nun Klar-/Geheimtextpaare auf der linken Seite einsetzten, können wir den Wert von $k_1 \oplus k_4 \oplus k_7 \oplus k_9$ vorhersagen. Damit haben wir die Anzahl der Bits die bei einem Brute-Force Angriff durchprobiert werden müssen um eins reduziert. ■

Ist p die Wahrscheinlichkeit, dass eine Beziehung richtig ist, so schreiben wir für den Bias

$$\varepsilon = p - \frac{1}{2}.$$

Wie schon erwähnt können wir durch Addition von 1 zu einer Gleichung immer ohne Beschränkung der Allgemeinheit $\varepsilon \geq 0$ voraussetzten.

Testet man nun n zufällige Paare und geht davon aus, dass diese Tests unabhängig sind, so liegt eine Binomialverteilung $Bi(n;p)$ vor. Die Wahrscheinlichkeit, dass mehr als die Hälfte den korrekten Wert haben ist dann

$$P = \sum_{k=\lceil (n+1)/2 \rceil}^{n} \binom{n}{k} p^k (1-p)^{n-k} \approx \Phi\left(\sqrt{n} \frac{p - \frac{1}{2}}{\sqrt{p(1-p)}} \right),$$

wobei wir die Binomialverteilung (nach dem zentralen Grenzwertsatz) für große n durch die Normalverteilung $\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} \mathrm{e}^{-x^2/2} dx$ genähert haben. Wollen wir, dass diese Wahrscheinlichkeit zumindest einen vorgegeben Wert P erreicht, so muss

$$\sqrt{n} \frac{p - \frac{1}{2}}{\sqrt{p(1-p)}} \geq z_P$$

gelten, wobei $z_P = \Phi^{-1}(P)$ das P-Quantil der Normalverteilung ist. Wir müssen also

$$n \geq z_P^2 \frac{p(1-p)}{(p-\frac{1}{2})^2} = z_P^2 \frac{1-4\varepsilon^2}{4\varepsilon^2}$$

wählen.

Haben wir insgesamt m Gleichungen, die jeweils mit Wahrscheinlichkeit P gültig sind, so ist die Wahrscheinlichkeit dass alle gütlig sind (wieder unter der Annahme dass diese Wahrscheinlichkeiten unabhängig sind) gleich P^m. Geben wir diese letze Wahrscheinlichkeit vor, so können wir daraus P bestimmen und daraus für jede zugehörige Gleichung die Anzahl der nötigen Paare.

Beispiel 4.17 Angenommen sie haben drei Gleichungen mit Bias $\varepsilon_1 = 0.1$, $\varepsilon_2 = 0.05$ und $\varepsilon_3 = 0.01$. Wie viele Klar-/Geheimtextpaare benötigen Sie, um die zugehörigen Schlüsselbits mit einer Wahrscheinlichkeit von zumindest 90% zu bestimmen? Wieviele Paare wären es bei 99%?

Lösung zu 4.17 Zunächst muss $P^3 = 0.9$ also $P = \sqrt[3]{0.9} = 0.965$. Das zugehörige Quantil ist $z_P = 1.82$. Also brauchen wir für die erste Gleichung $n_1 \approx 180$, für die zweite $n_2 \approx 730$ und für die dritte $n_2 \approx 18\,400$ Paare. Insgesamt sind also ca. 18 400 Paare nötig. Wegen $z_{0.99} = 2.71$ benötigt man etwas mehr als doppelt so viele Paare. ■

Bei mehr als zwei Runden wird die Sache natürlich komplizierter. Jetzt werden mehrere S-Boxen durchlaufen und wir müssen uns überlegen was dabei mit unseren Beziehungen passiert. Betrachten wir dazu zunächst zwei Zufallsbits X_1 und X_2 mit Bias $\varepsilon_j = P(X_j = 1) - \frac{1}{2}$. Unter der Annahme, dass beide Zufallsbits unabhängig

sind, berechnen wir

$$\begin{aligned} P(X_1 \oplus X_2 = 1) &= P(X_1 = 0, X_2 = 1) + P(X_1 = 1, X_2 = 0) \\ &= (\frac{1}{2} - \varepsilon_1)(\frac{1}{2} + \varepsilon_2) + (\frac{1}{2} + \varepsilon_1)(\frac{1}{2} - \varepsilon_2) = \frac{1}{2} + 2\varepsilon_1\varepsilon_2. \end{aligned}$$

Der Bias von $X_1 \oplus X_2$ ist also gleich $2\varepsilon_1\varepsilon_2$. Mit Induktion erhalten wir daraus für n Zufallsbits:

Satz 4.18 (Piling-up Lemma) Sind $X_1, \ldots, X_n$ unabhängige Zufallsbits mit Bias $\varepsilon_j = P(X_j = 1) - \frac{1}{2}$, so gilt

$$P(X_1 \oplus X_2 \oplus \cdots \oplus X_n = 1) = \frac{1}{2} + 2^{n-1}\varepsilon_1\varepsilon_2 \cdots \varepsilon_n.$$

Beachtenswert ist, dass ein Bit mit $|\varepsilon_j| = \frac{1}{2}$ keinen Enfluß hat (so wie es sein soll), aber ein einzelnes Bit mit Bias $\varepsilon_j = 0$ den Gesamtbias zu 0 macht!

Damit können wir nun das Durchlaufen mehrer S-Boxen verstehen. Angenommen wir haben zwei S-Boxen, die nacheinander durchlaufen werden. Der Eingang der ersten sei X_1, X_2 und der Ausgang Y_1, Y_2. Analog sei der Eingang der zweiten $\tilde{X}_1$, $\tilde{X}_2$ und der Ausgang $\tilde{Y}_1$, $\tilde{Y}_2$. Dazwischen wird permutiert: $Y_1 = \tilde{X}_2$, $Y_2 = \tilde{X}_1$. Nun brauchen wir zwei passende Beziehungen für die beiden S-Boxen. Z.B.

$$X_1 \oplus X_2 = Y_1, \qquad \tilde{X}_2 = \tilde{Y}_1 \oplus \tilde{Y}_2.$$

Mit passend meinen wir hier, dass die Ausgangsvariablen aus der ersten Beziehung mit den Eingangsvariablen der zweiten Beziehung zusammen passen, damit wir die erste, unter Verwendung von $Y_1 = \tilde{X}_2$, $Y_2 = \tilde{X}_1$ in die zweite einsetzten können:

$$\tilde{Y}_1 \oplus \tilde{Y}_2 = \tilde{X}_2 = Y_1 = X_1 \oplus X_2.$$

Damit haben wir eine Beziehung die den Eingang der ersten S-Box mit dem Ausgang der zweiten S-Box verbindet. Wir können diese Beziehung auch durch Addition der drei Beziehungen

$$(X_1 \oplus X_2 \oplus Y_1) \oplus (Y_1 \oplus \tilde{X}_2) \oplus (\tilde{X}_2 \oplus \tilde{Y}_1 \oplus \tilde{Y}_2) = 0 \oplus 0 \oplus 0 = 0$$

erhalten. Wenn wir nun annehmen, dass diese drei Teile unabhängig sind, so folgt aus dem Piling-up Lemma, dass der Bias der entstanden Beziehung gleich $2\varepsilon_1\varepsilon_2$ ist, wobei ε_1 bzw. ε_2 der Bias der ersten bzw. letzten Beziehung sind (die mittlere Beziehung ist ja immer richtig und trägt nichts bei). Die Unabhängigkeit wird zwar im Allgemeinen nicht exakt erfüllt sein, bei einem Verschlüsselungsalgorithmus, der für gute Konfusion und Diffusion sorgt, sollte diese Annahme aber zumindest näherungsweise gelten. In der Praxis ist das in der Regel auch so.

Zusammenfassend sieht der Angriff wie folgt aus: Suche anhand der Tabellen für die S-Boxen passende Beziehungen heraus, die miteinander wie eben beschrieben verknüpft werden können. Der Bias kann mithilfe des Piling-up Lemma geschätzt werden und sinkt schnell mit der Anzahl der Runden. Um so kleiner der Bias, umsomehr Klar-/Geheimtextpaare benötigt man.

Beispiel 4.19 (→CAS) Piling-up Lemma
Finden Sie eine lineare Beziehung für den S-DES mit 3 Runden.

Lösung zu 4.19 Nun werden in jedem Fall mindestens zwei S-Boxen durchlaufen und wir müssen daher mindestens zwei Beziehungen in einander einsetzten. Wir verwenden die gleiche Notation wie in Beispiel 4.16. Wir beginnen mit

$$Y^2 = P^{-1}(R(IP(y)) \oplus R(IP(x))$$

und verwenden die Beziehung

$$Y_3^2 \oplus X_5^2 \oplus X_6^2,$$

die laut Tabelle mit Bias -0.1875 gilt. Wegen

$$X^2 = E(L(IP(x)) \oplus P(Y^1)) \oplus k^2$$

folgt daraus

$$k_2 \oplus k_{10} \oplus x_3 \oplus x_5 \oplus x_6 \oplus Y_3^1 \oplus Y_4^1 \oplus y_5.$$

Ein Blick auf unsere Tabelle ergibt, dass

$$Y_3^1 \oplus Y_4^1 \oplus X_7^1$$

mit Bias 0.1875 gilt. Für X^1 gilt aber

$$X^1 = E(R(IP(x))) \oplus k^1$$

und somit

$$x_3 \oplus x_5 \oplus x_6 \oplus x_7 \oplus y_5 \oplus 1 = k_2,$$

wobei wir als Bias $2 \cdot 0.1875^2 = 0.0703125$ erwarten. Man kann mit dem Computer überprüfen, dass dieser Wert mit dem exakten Bias übereinstimmt. ■

Bei zu vielen Runden braucht man zu viele Paare und deshalb ist dieser Angriff beim DES nicht erfolgreich. Allgemein legt man zusätzlich beim Design der S-Boxen Wert darauf, dass es keine Beziehungen mit hohem Bias gibt und, dass beim Zusammensetzten der Beziehungen möglichst viele notwendig sind, damit die Rundenanzahl klein gehalten werden kann. Eine genauere Ausführung an einem Beispiel ist wieder in [Hay02] zu finden.

Versuchen wir am Ende nochmals die Erkenntnisse aus diesem Abschnitt zusammenzufassen. Zunächst ist festzuhalten, dass die hier beschriebenen Angriffe prinzipiell bei jedem Algorithmus der aus einer Hintereinanderausführung der Operationen Expansion, Permutation, Schlüsseladdition, S-Box (in beliebiger Reihenfolge) besteht. Praktisch alle modernen Verfahren sind von dieser Form.

Eine zentrale Rolle für die Sicherheit spielen die S-Boxen und sie müssen eine Reihe von Eigenschaften erfüllen:

- Sie sollten gute statistische Eigenschaften besitzten (Vollständigkeit bzw. annähernd das Lawienenkriterium erfüllen).
- Sie sollten keine linearen Beziehungen mit hohem Bias erfüllen.
- Sie sollten keine Eingangs-/Ausgangsdifferenzen mit hohen Wahrscheinlichkeiten haben.
- Sie sollten möglichst groß sein (dürfen aus praktischen Gründen aber nicht zu groß sein).
- Nicht-invertierbare S-Boxen haben den Vorteil, dass beim Zurückvervolgen mehrere Urbilder verfolgt werden müssen (in diesem Fall sollte es keine Ausgangswerte mit nur wenigen Urbildern geben), benötigen aber eine Feistel-Box.

Diese Liste darf aber nur als erster Hinweis verstanden werden. In der Literatur findet man eine Unzahl von Arbeiten die sich alleine dem Design von S-Boxen widmen.

Unsere Analyse zeigt uns auch die Schwachstellen des S-DES auf und wie sie verbessert werden können. Abgesehen natürlich von der zu kleinen Blockgröße und zu kleinen Schlüssellänge.

- Die Anzahl der Runden sollte möglichst groß sein (aber aus Gründen der Effizienz auch nicht unnötig groß).
- Wenn man die Schlüsselverknüpfung nicht mit XOR sondern nichtlinear (z.B mit AND oder OR) machen würde, dann wäre die differentielle Kryptoanalyse nicht mehr möglich und auch die lineare Kryptoanalyse würde erschwert. Allerdings verteilt nur XOR die Schlüsselinformationen gleichmäßig auf das Chiffrat, denn AND ergibt in drei von vier Fällen das Ergebnis 0 und OR in drei von vier Fällen 1.
- Man könnte in jeder Runde einen eigenen Schlüssel verwenden. Allerdings zielen viele Angriffe darauf ab, den Teilschlüssel der letzten Runde zu ermitteln. Ist das gelungen kann man die letzte Runde herausrechnen und die Runde davor angreiffen (bei einer nichtinvertierbaren S-Box steigt allerdings

die Anzahl der Möglichkeit die rückverfolgt werden müssen). Ein signifikanter Sicherheitsgewinn ist daher nicht zu erwarten.

Ausserdem könnte man bei der Teilschlüsselerzeugung einen nichtlinearen Bestandteil verwenden, damit Informationen über einen Teilschlüssel nicht sofort auch Informationen über die anderen Teilschlüssel liefern.

- Transformationen vor der ersten Schlüsseladdition bzw. nach der letzten können herausgerechnet werden und bringen aus kryptografischer Sicht nichts (eine nichtinvertierbare Transformation am Ende bringt einen kleinen Vorteil, da beim Zurückrechnen mehrere Fälle berücksichtigt werden müssen). Ebenso kann eine invertierbare lineare Transformation vor der letzten Schlüsseladdition leicht herausgerechnet werden. Aus diesem Grund verwendet man gerne als erste und als letzte Transformation eine Schlüsseladdition, was als **Key Whitening** bezeichnet wird.

Am Ende ist festzuhalten, dass die beschriebenen Angriffe natürlich auch kombiniert werden können. Neben der differentiellen und linearen Kryptoanlayse gibt es noch weitere Verfahren, die aber bislang noch keine durchschlagenden Erfolge vorweisen können.

4.3 Der Advanced Encryption Standard (AES)

Auch vor der Entwicklung des Spezialchip der Electronic Frontier Foundation war klar, dass die Tage von DES gezählt sind. Zwar bestand noch kein Anlass, an der Sicherheit von Triple DES zu zweifeln, doch ist er in Software ziemlich langsam. Deshalb wurde im Jänner 1997 vom NIST ein Aufruf zur Suche nach einem neuen Standard gestartet. Etwa drei Monate später wurden in einem öffentlichen Workshop die Anforderungen an den neuen „Advanced Encryption Standard“ (AES) festgelegt:

- AES musste eine symmetrische Blockchiffre mit mindestens 128 Bit Blocklänge sein und Schlüssel der Länge 128, 192 und 256 Bit zur Verfügung stellen.
- Er sollte sowohl in Software als auch in Hardware (Smartcards) effizient implementierbar sein.
- Er musste gegen alle bekannten Methoden der Kryptoanalyse (z.B.: lineare, differentielle) resistent sein. Wenn möglich, sollte auch eine mathematische Rechtfertigung der Sicherheit gegeben werden.
- Der Algorithmus musste frei von patentrechtlichen Ansprüchen sein und von jedermann kostenlos genutzt werden können.

Im August 1998 begann daraufhin die öffentliche Begutachtung der 15 eingereichten Kandidaten. Im März 1999 wurden aus diesen Kandidaten fünf ausgewählt, die in die engere Wahl kamen: **MARS**, **RC6**, **Rijndael**, **Serpent** und **Twofish**.

Auch nach eingehender Analyse dieser fünf Algorithmen durch Expert:innen aus aller Welt konnten bei keinem der Algorithmen Schwachstellen festgestellt werden. Jeder Algorithmus hat seine Vor- und Nachteile und jeder hätte der neue Standard werden können.

Deshalb wurde von einigen Expert:innen auch gefordert, *alle* Algorithmen als neuen Standard zu erklären. Gerade bei der Implementierung in Software ist es kein Problem, alle Verfahren zu implementieren und dem Anwender die Auswahl zu überlassen. Für jeden Algorithmus könnte es ein bevorzugtes Anwendungsgebiet geben (Smartcards, Online-Verschlüsselung, ...). Dies würde insgesamt die Flexibität erhöhen und Situationen wie im Bankenbereich vermeiden, wo man allein auf DES gesetzt hat und die Umstellung auf Triple DES dann Jahre gedauert und enorme Kosten verursacht hat. Außerdem besteht immer die theoretische Möglichkeit, dass bei einem der Algorithmen eine Schwachstelle gefunden wird, so dass dieser ersetzt werden muss.

Trotzdem hat sich das NIST im Sommer 2001 nur für Rijndael entschieden, der von den jungen belgischen Kryptologen Vincent **Rij**men (*1970) und Joan **Dae**men (*1965) entwickelt worden war, und ihn zum AES, dem neuen US-Verschlüsselungsstandard, gemacht. Die Gründe, warum nur ein Kandidat ausgewählt wurde, haben nicht alle überzeugt:

- Die geforderten Schlüssellängen bieten genügend Reserve, falls Rijndael doch irgendwann praktisch relevante Schwachstellen zeigen sollte. Im Notfall steht auch noch Triple DES als Alternative zur Verfügung.
- Es ist kostengünstiger, nur einen einzigen Algorithmus zu implementieren (das betrifft vor allem eine Hardware-Implementierung).
- Falls Erfinder ähnlicher Algorithmen patentrechtliche Ansprüche stellen sollten, so entstehen geringere Kosten.

Sehen wir uns nun den AES Algorithmus genauer an. Eine ausführliche Beschreibung finden Sie in [FIPS197]. AES ist viel einfacher zu überblicken als DES. Er ist ebenfalls ein Produktalgorithmus, d.h., jeder Klartextblock durchläuft mehrere Runden, die analog aufgebaut sind. Auch wird der Schlüssel in Teilschlüssel zerlegt, die in den einzelnen Runden verwendet werden. Im Gegensatz zum DES enthält der AES aber *keine* Feistel-Boxen. Erinnern wir uns: ihre Aufgabe beim DES war es, die Invertierbarkeit der nichtlinearen Transformation sicherzustellen. Die nichtlineare Transformation beim AES benötigt keine Feistel-Box, denn sie ist selbst invertierbar: Es handelt sich um die Berechnung des multiplikativen Inversen in $GF(2^8)$, d.h., die Abbildung eines Bytes x auf sein Inverses x^{-1}.

Die Transformation $f(x) = x^{-1}$ ist nichtlinear, denn: Eine lineare Abbildung f hat ja die Eigenschaft, dass $f(x+y) = f(x)+f(y)$, und das ist hier nicht der Fall: $f(x+y) = (x+y)^{-1}$ ist nicht dasselbe wie $f(x) + f(y) = x^{-1} + y^{-1}$. Und diese Abbildung ist invertierbar, sie ist sogar ihre eigene Umkehrabbildung: $f(f(x)) = x$.

Bei AES beträgt die Blocklänge 128 Bit, als Schlüssellänge stehen 128, 192 oder 256 Bit zur Wahl und man spricht von AES-128, AES-192 bzw. AES-256. Wir wollen den AES im Folgenden für eine Schlüssellänge von 128 Bit beschreiben (AES-128). Die Struktur ist aber immer die gleiche und es unterscheiden sich nur die Anzahl der Runden (10, 12 bzw. 14 bei AES-128, AES-192 bzw. AES-256) und die Details der Teilschlüsselerzeugung.

Ein Klartextblock enthält also 128 Bit: $b_0 b_1 \cdots b_{127}$. Diese werden zu 16 Bytes zusammengefasst, $s_{00} = b_0 \cdots b_7$, $s_{10} = b_8 \cdots b_{15}$, ..., $s_{33} = b_{120} \cdots b_{127}$, und als Spalten einer 4×4-Matrix geschrieben:

$$\begin{pmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{10} & s_{11} & s_{12} & s_{13} \\ s_{20} & s_{21} & s_{22} & s_{23} \\ s_{30} & s_{31} & s_{32} & s_{33} \end{pmatrix}$$

Jedes s_{ij} ist ein Byte.

Achtung: Die übliche Konvention ist es, die Bits eines Blocks von links beginnend durchzunummerieren, die Bits innerhalb eines Bytes aber von rechts beginnend.

Diese Matrix wird als **Zustand(smatrix)** (engl. *state*) bezeichnet. Sie durchläuft die zehn Runden und am Ende stehen die Geheimtextbytes in der Matrix und werden spaltenweise ausgelesen.

Vor Beginn der Runden werden aus dem 128-Bit-Schlüssel zehn Rundenschlüssel zu je 128 Bit erzeugt, und ebenfalls spaltenweise in 4×4-Matrizen geschrieben. Die Rundenschlüssel werden rekursiv erzeugt, anders als beim DES wird dabei eine nichtlineare Funktion (und nicht nur Permutationen) verwendet.

Die Runden sind (bis auf die letzte) immer gleich aufgebaut und bestehen aus vier Operationen:

- SubBytes; die (nichtlineare) S-Box des AES. Sie operiert bytewise auf der Zustandsmatrix und erzeugt keine ausreichende Diffusion. Deshalb folgt als nächstes
- ShiftRows und MixColumn; die (lineare) Diffusionsschicht des AES. Am Ende steht
- AddRoundKey; die Teilschlüsseladdition mit XOR.

Zu diesem prinzipiellen Ablauf gibt es zwei kleine Änderungen:

- Vor der ersten Runde wird bereits eine Schlüsseladdition mit dem Hauptschlüssel durchgeführt. Dadurch findet sowohl ganz am Anfang als auch ganz am Ende eine Schlüsseladdition statt, was als **Key Whitening** bezeichnet wird und auch bei anderen Algorithmen zum Einsatz kommt. Der Vorteil ist, dass ein Angreifer den Zustand vor der ersten Runde nicht mehr kennt, was die Komplexität für diverse Angriffe erhöht (ohne den Aufwand der Ver-/Entschlüsselung nennenswert zu erhöhen).

- In der letzten Runde wird die MixColumn-Transformation weggelassen. Sie kann bei einem Angriff herausgerechnet werden und bringt somit keine zusätzliche Sicherheit.

Zusammenfassend lautet die Struktur wie folgt:

$$y = AES_k(x) = (X_{k_{10}} \circ SR \circ S \circ R_{k_9} \circ \ldots \circ R_{k_1} \circ X_k)(x), \quad R_{k_j} = X_{k_j} \circ MC \circ SR \circ S,$$

wobei S die SubBytes-Transformation bezeichnet, SR und MC die ShiftRows- und MixColumn-Transformationen und X_k die Teilschlüsseladdition per XOR. Sehen wir uns nun die Bestandteile genauer an.

SubBytes

Diese Transformation ist der *nichtlineare* Bestandteil von AES. Sie wird für jedes Byte $a_7 \cdots a_1 a_0$ der Zustandsmatrix einzeln durchgeführt und besteht aus zwei Schritten:
1) Berechne zu $a_7 \cdots a_1 a_0$ das multiplikatives Inverses $b_7 \cdots b_1 b_0$ in $GF(2^8)$. Dabei ist der Modul $m(x) = x^8 + x^4 + x^3 + x + 1$ zu verwenden.

Das multiplikative Inverse wird mit dem erweiterten Euklidschen Algorithmus berechnet. Die Zuordnung zwischen Bytes und Polynomen erfolgt wie gewohnt gemäß $a_7 \cdots a_1 a_0 \mapsto a_7 x^7 + a_6 x^6 + \ldots + a_1 x + a_0$.

Das Byte 00000000 ist ein Sonderfall, da es ja kein multiplikatives Inverses besitzt. Das Problem wird aber behoben, indem man festlegt, dass es in Schritt 1) unverändert bleibt.
2) Danach folgt für jedes Byte $b_7 \cdots b_1 b_0$ eine *affine* Transformation:

$$\begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Das Ergebnis von Schritt 2) ist das Byte $b'_7 \cdots b'_1 b'_0$. Die konstante Matrix bzw. der konstante Vektor sind bei jeder AES Verschlüsselung gleich. Daher kann in der Praxis natürlich für jedes der $2^8 = 256$ Bytes das Ergebnis von SubBytes vorberechnet und in einer Tabelle (*substitution table*, siehe Figure 7 in [FIPS197]) abgespeichert werden. Daher kommt auch der Name „SubBytes".

Würde SubBytes nur die Abbildung auf das multiplikative Inverse enthalten, so wären gewisse Angriffe (z.B. Interpolationsangriffe) möglich. Die nachgeschaltete affine Transformation hat die Aufgabe, die Transformation komplexer zu machen und so diese Angriffe zu vereiteln. Insgesamt

ergibt sich bei Mehrfachausführung die Struktur eines Kettenbruchs. Bislang konnte das aber nicht für einen Angriff ausgenutzt werden.

Beispiel 4.20 (→CAS) SubBytes
Berechnen Sie für die Zustandsmatrix

$$\begin{pmatrix} 02 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \end{pmatrix}$$

das Ergebnis der SubBytes-Transformation. Verwenden Sie dabei, dass das multiplikative Inverse von x modulo $x^8 + x^4 + x^3 + x + 1$ gleich $x^7 + x^3 + x^2 + 1$ ist. Die Bytes der Zustandsmatrix sind hexadezimal angegeben.

Lösung zu 4.20 Wir führen für jedes Byte in der Zustandsmatrix (es kommen nur 02 und 00 vor), die beiden Schritte von SubBytes durch. Wir beginnen mit 02: Schritt 1) Die Hexadezimalzahl 02 lautet binär 00000010, dies entspricht dem Polynom x. Das zugehörige multiplikative Inverse ist laut Angabe $x^7 + x^3 + x^2 + 1$, was dem Byte 10001101, bzw. hexadezimal $8D$, entspricht.

In der Praxis wird, wie wir wissen, ein multiplikatives Inverses mit dem Erweiterten Euklidschen Algorithmus berechnet. Hier haben wir es einfachheitshalber als gegeben angenommen. Wir können aber zumindest die Probe machen: $x \cdot (x^7 + x^3 + x^2 + 1) = x^8 + x^4 + x^3 + x = 1 (\mathrm{mod}\, (x^8 + x^4 + x^3 + x + 1))$.

Schritt 2) Nun folgt die affine Transformation für $8D$. Dazu schreiben wir $8D$ wieder in binärer Form, $b_7 \cdots b_1 b_0 = 10001101$, und zwar als Spaltenvektor (mit b_0 als oberste Komponente und b_7 als unterste Komponente):

$$\begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix},$$

Den Ergebnisvektor schreiben wir wieder als Zeile: $b'_7 \cdots b'_1 b'_0 = 01110111$, was hexadezimal 77 entspricht.

Analog geht man für das Byte 00 (hexadezimal) bzw. 00000000 (binär geschrieben) vor: In Schritt 1) bleibt es per Definition unverändert, da es ja kein

multiplikatives Inverses besitzt. In Schritt 2) erhalten wir

$$\begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix},$$

also $b'_7 \cdots b'_1 b'_0 = 01100011$ bzw., hexadezimal, 63.
Der Zustand nach SubBytes ist damit

$$\begin{pmatrix} 77 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \end{pmatrix}.$$

■

ShiftRows

Hier werden die einzelnen Zeilen des Zustands

$$\begin{pmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{10} & s_{11} & s_{12} & s_{13} \\ s_{20} & s_{21} & s_{22} & s_{23} \\ s_{30} & s_{31} & s_{32} & s_{33} \end{pmatrix}$$

zyklisch nach links rotiert, und zwar die erste Zeile um 0, die zweite um 1, die dritte um 2 und die vierte um 3 Stellen. Das Ergebnis von ShiftRows ist somit:

$$\begin{pmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{11} & s_{12} & s_{13} & s_{10} \\ s_{22} & s_{23} & s_{20} & s_{21} \\ s_{33} & s_{30} & s_{31} & s_{32} \end{pmatrix}.$$

Beispiel 4.21 ShiftRows
Berechnen Sie für die Zustandsmatrix

$$\begin{pmatrix} AB & D5 & 04 & 88 \\ AB & D5 & 04 & 88 \\ AB & D5 & 04 & 88 \\ AB & D5 & 04 & 88 \end{pmatrix}$$

das Ergebnis der ShiftRows-Transformation. (Die Bytes der Zustandsmatrix sind hexadezimal angegeben.)

Lösung zu 4.21 Die erste Zeile bleibt gleich, die folgenden Zeilen werden um eine, zwei bzw. drei Stellen zyklisch nach links verschoben:

$$\begin{pmatrix} AB & D5 & 04 & 88 \\ D5 & 04 & 88 & AB \\ 04 & 88 & AB & D5 \\ 88 & AB & D5 & 04 \end{pmatrix}$$

■

MixColumns

Bei dieser Transformation wird jede Spalte des Zustands

$$\begin{pmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{10} & s_{11} & s_{12} & s_{13} \\ s_{20} & s_{21} & s_{22} & s_{23} \\ s_{30} & s_{31} & s_{32} & s_{33} \end{pmatrix}$$

mit einer fix gegebenen Matrix multipliziert:

$$\begin{pmatrix} s'_{0i} \\ s'_{1i} \\ s'_{2i} \\ s'_{3i} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} s_{0i} \\ s_{1i} \\ s_{2i} \\ s_{3i} \end{pmatrix}.$$

Die sich ergebenden Spalten bilden die Spalten des Zustands nach MixColumns.

$$\begin{pmatrix} s'_{00} & s'_{01} & s'_{02} & s'_{03} \\ s'_{10} & s'_{11} & s'_{12} & s'_{13} \\ s'_{20} & s'_{21} & s'_{22} & s'_{23} \\ s'_{30} & s'_{31} & s'_{32} & s'_{33} \end{pmatrix}$$

Da die Matrixelemente bzw. die Elemente der Spalten der Zustandsmatrix Bytes sind (wieder hexadezimal angegeben), ist die Multiplikation in $GF(2^8)$ mit dem Modul $m(x) = x^8 + x^4 + x^3 + x + 1$ durchzuführen.

Beispiel 4.22 MixColumns
Berechnen Sie für die Zustandsmatrix

$$\begin{pmatrix} 00 & 00 & 00 & 00 \\ 01 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 02 & 00 & 00 & 00 \end{pmatrix}$$

das Ergebnis der MixColumns-Transformation. (Die Bytes der Zustandsmatrix sind wieder hexadezimal angegeben.)

Lösung zu 4.22 Für die erste Spalte ergibt sich zunächst (ganz normale Matrixmultiplikation):

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} 00 \\ 01 \\ 00 \\ 02 \end{pmatrix} = \begin{pmatrix} 01 \cdot 03 + 02 \cdot 01 \\ 01 \cdot 02 + 02 \cdot 01 \\ 01 \cdot 01 + 02 \cdot 03 \\ 01 \cdot 01 + 02 \cdot 02 \end{pmatrix}$$

Nun sind die einzelnen Bytes zu multiplizieren bzw. zu addieren: Dazu werden sie zunächst als Polynome geschrieben, $01 = 1$, $02 = x$, $03 = x+1$, dann werden diese in $GF(2^8) = \mathbb{Z}_2[x]_{x^8+x^4+x^3+x+1}$ multipliziert bzw. addiert:

$$\begin{pmatrix} 01 \cdot 03 + 02 \cdot 01 \\ 01 \cdot 02 + 02 \cdot 01 \\ 01 \cdot 01 + 02 \cdot 03 \\ 01 \cdot 01 + 02 \cdot 02 \end{pmatrix} = \begin{pmatrix} 1 \cdot (x+1) + x \cdot 1 \\ 1 \cdot x + x \cdot 1 \\ 1 \cdot 1 + x \cdot (x+1) \\ 1 \cdot 1 + x \cdot x \end{pmatrix} = \begin{pmatrix} x + 1 + x \\ x + x \\ 1 + x^2 + x \\ 1 + x^2 \end{pmatrix} = \begin{pmatrix} 01 \\ 00 \\ 07 \\ 05 \end{pmatrix}.$$

Bei diesem Beispiel entsteht kein Polynom mit Grad größer 7, das Ergebnis ist also immer wieder ein Byte. Würde ein Polynom mit Grad größer 7 entstehen, so müsste, um wieder ein Byte zu erhalten, der Rest modulo $x^8 + x^4 + x^3 + x + 1$ genommen werden.

Die übrigen Spalten werden auf die Spalte mit lauter 00 abgebildet, also erhalten wir insgesamt als Ergebnis der MixColumns-Transformation:

$$\begin{pmatrix} 01 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 07 & 00 & 00 & 00 \\ 05 & 00 & 00 & 00 \end{pmatrix}.$$

■

Man kann sich überlegen, dass die MixColumns-Transformation der Multiplikation jeder Spalte (geschrieben als Polynom mit Bytes als Koeffizienten, also $s_{3i}x^3 + s_{2i}x^2 + s_{1i}x + s_{0i}$) mit dem festen Polynom $03x^3+01x^2+01x+02$ entspricht; dabei ist das Produkt modulo $m(x) = 01x^4+01$ zu nehmen, damit das Ergebnis wieder ein Polynom vom Grad ≤ 3 ist, also die Form $b_{3i}x^3 + b_{2i}x^2 + b_{1i}x + b_{0i}$ hat. Es wird also in $GF(2^8)[x]_{01x^4+01}$ gerechnet. (Das feste Polynom $03x^3 + 01x^2 + 01x + 02$ ist teilerfremd zu $01x^4 + 01$; diese Eigenschaft ist notwendig, da ansonsten diese Transformation nicht umkehrbar wäre).

AddRoundKey

Hier wird der Zustand per XOR mit dem Rundenschlüssel verknüpft.

Teilschlüsselerzeugung

Bei AES-128 werden aus dem 128-Bit-Schlüssel insgesamt 10 Rundenschlüssel mit je 128-Bit Länge erzeugt.

Dazu wird der Schlüssel zunächst in 4 Teile („Wörter“) zu je 4 Bytes zerlegt. Diese werden dann rekursiv untereinander verknüpft (mit XOR, Permutationen und SubBytes) und sukzessive für jede Runde 4 Wörter zu je 4 Bytes abgeleitet, die dann zusammengesetzt den jeweiligen Rundenschlüssel bilden. Aufgrund der Verwendung von SubBytes ist insbesondere auch bei der Teilschlüsselerzeugung ein nichtlinearer Anteil vorhanden. Für Details siehe Abschnitt 5.2 „Key Expansion“ bzw. Appendix A.1 (konkretes Beispiel) in [FIPS197] oder auch [DR01].

Entschlüsselung

Da SubBytes, ShiftRows, MixColumns und AddRoundKey umkehrbar sind, müssen zur Entschlüsselung nur die inversen Transformationen in umgekehrter Reihenfolge angewandt werden.

Sicherheit

Die Transformationen SubBytes, ShiftRows und MixColumns sind so gewählt, dass sie möglichst einfach und übersichtlich sind (und damit gut zu kryptoanalysieren). Es gibt inzwischen zwar einige Ansätze für Angriffe, aber keiner davon stellt eine praktische Bedrohung dar. Wie beim DES wird jedoch erst durch die fortgesetzte Hintereinanderausführung der Runden Sicherheit erzeugt. Die Sicherheit steigt dabei drastisch mit der Anzahl der durchlaufenen Runden. Die Designkriterien und viele weitere Informationen sind in [DR01] nachzulesen.

Am Ende sei noch erwähnt, dass der AES zwar aktuell der wichtigste symmetrische Algorithmus ist, aber natürlich nicht der einzige. Es gibt auch noch weitere Algorithmen, die für spezielle Einsatzbereiche optimiert sind. Es soll hier also nicht der Eindruck entstehen, dass der AES in jeder Situation die beste Wahl ist.

4.4 Betriebsarten

Während eine Blockchiffre nur jeweils einen Klartextblock in einen Geheimtextblock umwandeln kann, legt zusätzlich die **Betriebsart** (**Betriebsmodus**, engl. *block cipher mode*) fest, wie eine *Folge* von Blöcken zu verschlüsseln ist.

Es gibt fünf Standardbetriebsmodi, die zum Zweck der Verschlüsselung mit einer beliebigen symmetrischen Blockchiffre (z.B. AES) verwendet werden können: Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), and Counter (CTR) [NIST01]. Darüber hinaus

gibt es weitere Betriebsmodi, die neben Verschlüsselung auch Nachrichtenintegrität sicherstellen, z.B. den Galois Counter Mode (GCM).

Bei einem Teil dieser Betriebsarten ist es notwendig, dass der Klartext als Folge von vollständigen Blöcken vorliegt. Wenn der letzte Block also nicht lang genug ist, muss er zuvor geeignet aufgefüllt werden (**Padding** = Hinzufügen von Leerinformation); dabei vergrößert sich die Datenmenge, was gerade bei kurzen Nachrichten ins Gewicht fallen kann.

Eine einfache Möglichkeit des Paddings ist es (**Bit padding**), am Ende ein einzelnes Bit 1 anzuhängen und dann mit weiteren Nullen aufzufüllen. So erkennt der Empfänger den Padding-Teil. (Da der Empfänger im Allgemeinen nicht feststellen kann, ob ein Padding notwendig war oder nicht, muss im Fall eines von vornherein vollständigen letzten Klartextblocks trotzdem ein zusätzlicher Padding-Block $100\cdots0$ angehängt werden. Die Regel für den Empfänger ist dann, beim letzten Block alle Bits ab der letzten 1 zu entfernt. Da dieser Fall regelmäß auftritt und dann immer zum gleichen Block führt, erleichtert diese einfache Methode einen Klartextangriff.)

Bei der folgenden Beschreibung der Betriebsmodi bezeichnet x_j immer einen Klartextblock, y_j den zugehörigen Geheimtextblock und E den Blockalgorithmus inklusive verwendetem geheimen Schlüssel k (z.B. AES). Der Schlüssel wird aber übersichtlichkeitshalber nicht extra angeführt.

Electronic Codebook Mode (ECB): Diese Betriebsart kennen wir bereits – jeder Klartextblock wird für sich verschlüsselt:

$$y_j = E(x_j).$$

Entschlüsselt wird mit

$$x_j = E^{-1}(y_j).$$

Vor- und Nachteile:

- Padding notwendig.
- Gleiche Klartextblöcke ergeben gleiche Geheimtextblöcke und das ermöglicht eine sogenannte **Verkehrsanalyse**: Ein Angreifer erkennt, wenn die gleiche Nachricht übertragen wurde bzw. erhält unter Umständen gewisse Informationen über die Struktur des Klartextes. Ev. kann er nach einiger Zeit die Bedeutung einzelner Blöcke erkennen (indem er seine Beobachtungen mit anderen Informationen verküpft oder bewusst Nachrichten provoziert). Im schlimmsten Fall gelingt es ihm in die Kommunikation einzugreifen und Blöcke auszutauschen. Nehmen wir z.B. an, Sie belauschen die Kommunikation ihrer Bank und sind irgendwann in der Lage Transaktionen zu erkennen, die ihrem Konto etwas gutschreiben. Dann speichern Sie eine solche Nachricht und senden sie am nächsten Tag nochmals. Schon bekommen Sie den Betrag zweimal gutgeschrieben. In Abbildung 4.5 ist das nochmals veranschaulicht. Dabei wurde ein Bild mit einer einfachen Vigenère-Verschlüsselung (die Pixel wurden zu Bytes zusammengefasst und das Schlüsselwort wurde nach

ASCII-Code mittels Xor verknüpft). Auch bei einem langen Schlüsselwort (was einer langen Blockgröße entspricht), ist die Struktur des Originalbildes noch gut erkennbar. Insbesondere hängt der Effekt von der Blockgröße (und nicht von der Qualität der Verschlüsselungsfunktion) ab.

Eine klassische Gegenmaßnahme ist es, die Nachricht um zufällige Stücke zu erweitern (die der Empfänger wieder entfernt), was ebenfalls als Padding bezeichnet wird.

- Die Fehlerausbreitung ist sehr begrenzt: ein falsch übertragenes Bit im Block y_k bewirkt nur eine fehlerhafte Entschlüsselung des Blockes x_k.
- Blöcke können parallel ver-/entschlüsselt werden.

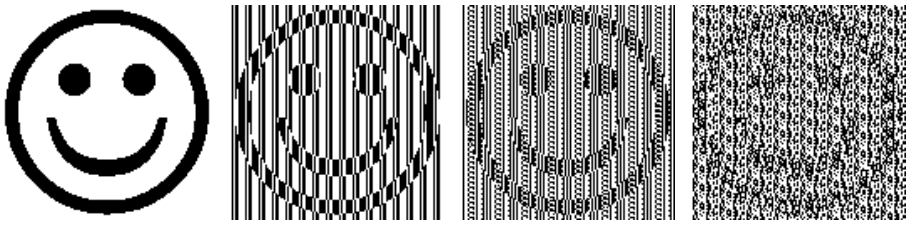

Abbildung 4.5: Originalbild (links) mit Vigenére im ECB-Modus und mit verschieden Schlüsselwörter verschlüsselt: „X“, „GEHEIM“, „Stre4gG3he1mesPA$$W0RT“

Cipher Block Chaining Mode (CBC): Vor der Verschlüsselung wird ein Klartextblock nun mit dem zuvor erzeugten Geheimtextblock mittels XOR verknüpft. Da es für den ersten Klartextblock keinen vorherigen Geheimtextblock gibt, muss ein Initialisierungsvektor IV gewählt werden:

$$y_0 = IV, \qquad y_j = E(x_j \oplus y_{j-1}).$$

Entschlüsselt wird mit

$$x_j = E^{-1}(y_j) \oplus y_{j-1}.$$

Der Initialisierungsvektor IV wird als 0-ter Geheimtextblock mitübermittelt. Das stellt kein Sicherheitsrisiko dar, denn ein Angreifer kann mit y_0 ja nichts anfangen.

- (Kryptographisch sicheres) Padding notwendig (ansonsten ist ein **Padding-Oracle-Attack** möglich).
- Jeder Geheimtextblock hängt nun von allen vorangehenden Klartextblöcken ab und somit werden Klartextmuster zerstört (Diffusion). Das ist in Abbildung 4.6 veranschaulicht. Während beim ECB-Modus (2. Bild) die Struktur des Originals (1. Bild) noch zu erkennen ist, ist das beim OFB-Modus (3. Bild) nicht mehr der Fall.

- Werden verschiedene Initialisierungsvektoren (z.B. Zeitstempel, Zufallsfolge) gewählt, so werden gleiche Klartexte in verschiedene Geheimtexte übergeführt.

- Findet man im Geheimtext eine **Kollision**, $y_j = y_k$, so gilt $x_j \oplus x_k = E^{-1}(y_j) \oplus y_{j-1} \oplus E^{-1}(y_k) \oplus y_{k-1} = y_{j-1} \oplus y_{k-1}$ und das kann für einen Angriff genutzt werden. Kollisionen findet man bei $O(2^{n/2})$ Geheimtexten (Geburstagsangriff, den wir noch besprechen werden). Das wird zum Problem bei zu kleiner Blockgröße (z.B. die Verwendung von Algorithmen mit nur 64 Bit Blöcken (Triple-DES, Blowfish) in gängigen Internetanwendungen wie IPSec oder HTTPS: **Sweet32-Angriff**).

- Da man zum Entschlüsseln nur den aktuellen und den vorhergehenden Block benötigt, können die Blöcke in beliebiger Reihenfolge entschlüsselt werden (notwendig bei der Verschlüsselung von Datenträgern, wenn man nur auf einen Teil der Daten zugreifen möchte). Insbesondere können Blöcke parallel entschlüsselt werden.

- Begrenzte Fehlerausbreitung: Ein falsch übertragenes Bit im Block y_k bewirkt fehlerhafte Entschlüsselung der beiden aufeinanderfolgenden Blöcke $x_k = E^{-1}(y_k) \oplus y_{k-1}$ und $x_{k+1} = E^{-1}(y_{k+1}) \oplus y_k$.

- In [NIST01] gibt es auch eine Variante für den Fall, in dem der Klartext in Blöcken vorliegt, die kleiner sind als die Blockgröße der verwendeten Blockchiffre.

Cipher Feedback Mode (CFB): Bei dieser Betriebsart wird der im vorigen Schritt entstandene Geheimtextblock verschlüsselt und dann mit dem aktuellen Klartextblock verknüpft:

$$y_0 = IV, \qquad y_j = E(y_{j-1}) \oplus x_j.$$

Entschlüsselt wird mit

$$x_j = y_j \oplus E(y_{j-1}).$$

- Vorteil ist, dass zur Verschlüsselung eines Klartextblockes x_j nun nicht der *ganze* Block vorliegen muss. Sobald das erste Bit von x_j vorliegt, kann man es auch schon verschlüsseln und an den Empfänger senden. Es können somit Datenströme verschlüsselt werden, ohne dass bei Teilblöcken gewartet oder aufgefüllt werden muss. Man kann somit die Blockchiffre auch als **Stromchiffre** betreiben.

- Gleichen Vorteile beim Entschlüsseln wie bei CBC.

- Gleiche Klartextblöcke ergeben verschiedene Geheimtextblöcke (Klartextmuster werden zerstört); bei verschiedenen IVs werden gleiche Klartexte in verschiedene Geheimtexte übergeführt. Vergleiche wiederum Abbildung 4.6.

- Der Schlüsselstrom $z_1, z_2, \ldots = E(y_0), E(y_1) \ldots$ ist vom Klartext abhängig. Dadurch wird die Verschlüsselung sicherer, aber der Schlüsselstrom kann nicht wie bei OFB im Vorhinein berechnet werden.

- Begrenzte Fehlerausbreitung: Ein falsch übertragenes Bit bei y_k bewirkt fehlerhafte Entschlüsselung der aufeinanderfolgenden Blöcke $x_k = y_k \oplus E(y_{k-1})$ und $x_{k+1} = y_{k+1} \oplus E(y_k)$.

Output Feedback Mode (OFB): Diese Betriebsart ist ähnlich wie CFB, nur wird nun E quasi als Pseudozufallszahlengenerator für ein „One-Time-Pad“ verwendet:

$$y_j = z_j \oplus x_j, \qquad z_0 = IV, \quad z_j = E(z_{j-1}).$$

Entschlüsselt wird mit

$$x_j = y_j \oplus z_j, \qquad z_0 = IV, \quad z_j = E(z_{j-1}).$$

- Achtung, der IV (und der gleiche Schlüssel) darf niemals mehrfach verwendet werden: Kennt ein Angreifer einen weiteren Geheimtext $\tilde{y}_j = z_j \oplus \tilde{x}_j$, bei dem der gleiche Schlüssel verwendet wurde, so kann er aus $\tilde{y}_j \oplus y_j = \tilde{x}_j \oplus x_j$ Informationen über die Struktur der Klartexte erhalten (siehe Abbildung 2.3 für eine grafische Veranschaulichung). Noch schlimmer, gelangt der Angreifer an einen der beiden Klartexte, so kann er sofort auch den anderen berechnen. Dieser Fehler wurde dem WEP (802.11) Protokoll zur Sicherung von Funknetzten (WIFI) zum Verhängnis [BGW01].

- Es kann verschlüsselt werden, sobald das erste Bit eines Klartextblockes vorliegt (**Stromchiffre**). Da die Folge z_j unabhängig vom Klartext ist, kann sie im Voraus berechnet werden.

- Gleiche Klartextblöcke ergeben wieder verschiedene Geheimtextblöcke (Klartextmuster werden zerstört). Abhängig vom Algorithmus könnte es auch passieren, dass bestimmte IV eine Folge von z_j ergeben, die sich nach wenigen Blöcken periodisch wiederholt (*short cycle*). Das ist in Abbildung 4.6 passiert, wo beim 5. Bild die Struktur des Originals noch zu erahnen ist.

- Kennt ein Angreifer den Klartext und kann in die Kommunikation eingreifen, so könnte er mittels $\tilde{y}_j = y_j \oplus x_j \oplus \tilde{x}_j$ die übertragene Nachricht ändern (*bit-flipping attack*).

- Keine Fehlerausbreitung: Ein falsch übertragenes Bit bei y_k bewirkt fehlerhafte Entschlüsselung von $x_k = y_k \oplus z_k$ nur an derselben Bitposition.

Counter Mode (CTR): Diese Betriebsart ist ähnlich wie OFB, der Unterschied liegt im Pseudozufallszahlengenerator. Dieser enthält keine Rückkopplung, sondern eine Folge von sogenannten Zählern (*counters*) $t_1, t_2, \ldots, t_n$, die zuvor geeignet zu

erzeugen sind (z.B. Zahlen t_j, die aus zwei Blöcken $t_j = IV||CTR_j$ bestehen: der erste Block IV wird als Initialisierungsvektor zufällig gewählt und der zweite Block CTR_j wird in jedem Schritt um Eins erhöht):

$$y_j = z_j \oplus x_j, \qquad z_j = E(t_j).$$

Entschlüsselt wird mit

$$x_j = y_j \oplus z_j, \qquad z_j = E(t_j).$$

- Die Vor-/Nachteile ergeben sich wie beim OFB. Der Schlüsselstrom $z_j = E(t_j)$ ist wiederum nicht vom Klartext abhängig und kann daher im Vorhinein berechnet werden. Mehr noch, ein Schlüsselblock hängt nun nicht vom vorhergehenden Schlüsselblock ab, es können also Teile des Schlüssels unabhängig voneinander berechnet werden und somit auch Blöcke unabhängig voneinander entschlüsselt werden (praktisch für die Verschlüsselung von Datenträgern). Wie beim OFB darf jeder Zähler t_j nur einmal verwendet werden (besteht der Zähler t_j wie oben beschrieben aus zwei Teilen, $t_j = IV||CTR_j$, so wird das durch den zweiten Teil CTR_j sichergestellt). Dadurch kann es auch zu keinen Folgen z_j mit kleiner Periode wie bei OBF kommen.

- Blöcke können parallel ver-/entschlüsselt werden.

- Es gibt auch eine Variante, den **Galois Counter Mode** (**GCM**), bei der gleichzeitig eine kryptographishce Prüfziffer über alle Blöcke gebildet wird. Wir werden diese am Ende von Abschnitt 4.5 besprechen.

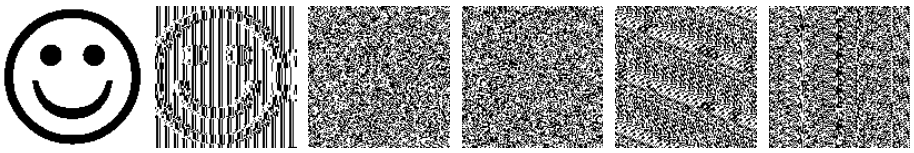

Abbildung 4.6: Originalbild (links) mit S-DES und verschiedenen Betriebsmodi verschlüsselt: ECB, CBC, CFB, OFB, CTR

Bei den letzten drei Modi (CFB, OFB, CTR) ist es übrigens zur Entschlüsselung nicht notwendig, E zu invertieren. Man hätte also beim Design von E etwas mehr Freiheiten als bei einer normalen Blockchiffre, könnte also für E auch eine kryptographische Hashfunktion verwenden.

Da alle Modi ihre Vor- und Nachteile haben, hängt es von der konkreten Anwendung ab, welcher am besten zu wählen ist.

Außerdem gehen die oben beschriebenen Modi nicht auf das Problem ein, dass der Angreifer in die Kommunikation eingreifen könnte. In diesem Fall könnte er z.B. Blöcke entfernen und dadurch Schaden anrichten oder z.B. durch das erneute Senden abgehörter Nachrichten Verwirrung

stiften (bzw. aus der Reaktion des Empfängers Rückschlüsse auf den Inhalt ziehen). Um das zu verhindern, könnte man die Nachrichten z.B. mit einem Zeitstempel und einer Prüfziffer versehen. Dann kann der Empfänger feststellen, wenn ein Angreifer in die Kommunikation eingreift (siehe MACs im nächsten Abschnitt).

4.5 Kryptographische Hashfunktionen

Sie kennen **Hashfunktionen** vielleicht als Hilfsmittel für effektive Speicherverfahren, die das schnelle Auffinden von Datensätzen ermöglichen. Hashfunktionen können auch als Prüfziffern eingesetzt werden.

Eine Hashfunktion h ist eine Funktion, die Daten beliebiger Länge auf Daten der festen Länge n Bit abbildet. Man nennt $h(x)$ den **Hashwert** von x. Wenn zwei verschiedene Daten x_1 und x_2 denselben Hashwert haben, so spricht man von einer **Kollision**. Da es unendlich viele Datensätze beliebiger Länge gibt, aber nur 2^n Bitfolgen der Länge n, kann eine Hashfunktion nie injektiv sein. Kollisionen sind daher *unvermeidbar*.

Wenn wir z.B. alle Daten der Länge 2^{23} Bit (1MB) betrachten, und für den Hashwert $128 = 2^7$ Bit wählen, dann fallen im besten Fall, wenn die Daten gleichmäßig auf die Hashwerte verteilt sind, auf jeden Hashwert $2^{16} = 65\,536$ Datensätze!

In der Kryptographie werden sogenannte **kryptographische Hashfunktionen** verwendet (siehe [NIST12]). Eine Mindestanforderung dafür ist die

- **Einweg-Eigenschaft** (**Urbildresistenz**, engl. **preimage resistance**); wenn es *praktisch* unmöglich ist, vom Hashwert h auf (irgend-) eine Nachricht x zurückzuschließen (es gibt ja wegen der Nicht-Injektivität immer mehrere), die diesen Hashwert besitzt.

Urbildresistenz ist z.B. wichtig bei der Verwendung von kryptographischen Hashfunktionen zur **Passwortverschlüsselung**. Passwörter werden dabei mittels Einweg-Hashfunktion verschlüsselt in einer Passwort-Datei abgelegt. Somit kann auch ein Systemadministrator die Passwörter nicht lesen. Bei Eingabe eines Passwortes durch den Nutzer wird jeweils der Hashwert berechnet und dieser mit dem in der Passwort-Datei gespeicherten Wert verglichen (mehr dazu im nächsten Abschnitt). Für einen Angreifer ist es wegen der Einweg-Eigenschaft dann praktisch nicht möglich, vom Hashwert auf das Passwort (bzw. irgendein Passwort mit diesem Hashwert) rückzuschließen **Urbildangriff** (**preimage attack**).

Zusätzlich zur Einwegeigenschaft wird bei kryptographischen Hashfunktionen **Kollisionsresistenz** gefordert, wobei es folgende Abstufung gibt: Eine Hashfunktion heißt

- **schwach kollisionsresistent** (engl. **(2nd) preimage resistance**), wenn es praktisch nicht möglich ist, zu einer *gegebenen* Nachricht x eine zweite Nachricht $x' \neq x$ zu finden, die denselben Hashwert besitzt.

- **(stark) kollisionsresistent** (engl. **collision resistance**), wenn es praktisch nicht möglich ist, *beliebige* Nachrichten x_1, x_2 mit $x_1 \neq x_2$ zu finden, die denselben Hashwert besitzen.

Die (schwache) Kollisionsresistenz ermöglicht z.B. die Anwendung als **Prüfziffer** (**digitaler Fingerabdruck**), die die Unversehrtheit von Dokumenten (bzw. Nachrichten) gewährleisten soll: Es ist für einen Angreifer praktisch nicht möglich, ein Dokument gezielt zu ändern, ohne den Hashwert zu ändern.

Als Prüfziffer war lange der **MD5-Algorithmus (Message Digest Version 5)** in Verwendung. Er wurde von Ronald Rivest, einem der Erfinder des RSA-Algorithmus, entwickelt und berechnet aus einer beliebigen Nachricht x einen 128-Bit Hashwert. MD5 gilt nicht mehr als sicher, soll aber als grundlegendes Beispiel kurz besprochen werden:

- Zunächst wird die Nachricht x mit vorgegebenen Bits aufgefüllt (padding), um eine Länge, die ein Vielfaches von 512 Bit ist, zu erreichen. Nun kann die Nachricht in m Blöcke der Länge 512 zerlegt werden: $x = (x_1, \ldots, x_m)$.

- Nun werden mit einer fixen Initialisierung h_0 die Werte h_j rekursiv gemäß

$$h_j = h_{j-1} \oplus f(h_{j-1} \| x_j)$$

  berechnet, solange, bis kein Nachrichtenblock mehr übrig ist. Der letzte Wert h_m ist der Hashwert der Nachricht x. Hier bezeichnet $\|$ die Aneinanderreihung (Konkatenation) der Blöcke.

  Dieses Schema ist als **Merkle-Damgård-Konstruktion** bekannt (nach den Kryptologen Ralph Merkle (*1952) und Ivan Damgård (*1956)). Dabei muss f eine kollissionsresistente Kompressionsfunktion (die Eingabe ist länger als die Ausgabe) sein. Z.B. könnte man eine Blockverschlüsselung mit x als Schlüssel verwenden: $f(h, x) = E_x(h)$. Passen die Blocklängen der verwendeten Algorithmen nicht zusammen, muss gegebenfalls eine Expansions- oder Kompressionsfunktion verwendet werden. Das zusätzliche xor mit h_{j-1} dient in diesem Fall dazu die Invertierbarkeit zu zerstören. Analog wie bei DES und AES wird bei f auf Diffusion (lineares Durchmischen) und Konfusion (eine nichtlineare Operation) in mehreren Runden gesetzt. Für Details sei auf [RFC1321] verwiesen.

Kryptographische Hashfunktionen werden auch bei **digitalen Signaturen** verwendet. Bei einer digitalen Signatur wird nur der kryptographische Hashwert und nicht das ganze Dokument unterschrieben. (Denn bei langen Nachrichten wäre es ineffizient, die gesamte Nachricht zu signieren.) Man spricht auch vom **kryptographischen Fingerabdruck**, der das Dokument hier repräsentiert. Die digitale Unterschrift ist dann für jedes Dokument, das denselben Hashwert wie das Original hat, gültig. Schwache bzw. starke Kollisionsresistenz macht es für einen Angreifer

praktisch unmöglich, das signierte Dokument zu ändern, wie die folgenden Überlegungen zeigen:

Stellen wir uns vor, dass ein Kaufvertrag x digital signiert wurde, genau genommen der kryptographische Hashwert $h = h(x)$ des Kaufvertrages. Die erstellte Signatur ist also grundsätzlich für jedes Dokument gültig, das denselben Hashwert h besitzt. Diese Tatsache möchte ein Angreifer Mallory nutzen:

Angriff auf schwache Kollisionsresistenz (**Zweites Urbild Angriff, 2nd preimage attack**): Mallory besitzt den (unterschriebenen) Kaufvertrag x, der den Hashwert h hat. Mallory ändert zunächst die Kaufsumme im Vertrag auf einen neuen für ihn günstigen Wert (dabei ändert sich natürlich der Hashwert) und dann versucht er, so lange unauffällige Änderungen (z.B., Einfügen von Leerzeichen, kleinere Umformulierungen, etc.) durchzuführen, bis sich wieder der ursprüngliche Hashwert ergibt. Mit dem gefälschten Vertrag x' kann Mallory nun zur Bank des Käufers gehen und sich die höhere Kaufsumme ausbezahlen lassen. Da der Hashwert und damit die Signatur gültig (unverändert) ist, wird der Betrug von der Bank nicht erkannt.

Wie groß ist der Aufwand, um diesen Angriff durchzuführen? Gehen wir davon aus, dass die $N = 2^n$ möglichen Hashwerte gleichmäßig verteilt sind. Ist nun ein konkreter Hashwert h gewünscht, so ist die Wahrscheinlichkeit, dass eine zufällig ausgewählte Datei genau diesen Hashwert hat, gleich $p = N^{-1} = 2^{-n}$. Wählt Mallory zufällig k Dateien, so ist die Wahrscheinlichkeit dafür, dass mindestens eine darunter den gewünschten Hashwert h hat, gleich

$$1 - (1 - \frac{1}{N})^k.$$

Das ist die Gegenwahrscheinlichkeit zum Ereignis „k Misserfolge“.

Damit diese Wahrscheinlichkeit zumindest 50% wird müssen

$$k = \frac{\ln(1/2)}{\ln(1 - N^{-1})} \approx \ln(2)N \approx 2^n$$

Dateien probiert werden.

Um das zu sehen, muss man die Gleichung $1 - (1 - N^{-1})^k = 1/2$, also $1/2 = (1 - N^{-1})^k$ nach k auflösen. Wir erhalten dann $k = \frac{\ln(1/2)}{\ln(1-N^{-1})}$. Danach wird noch die Näherung $\ln(1+x) = x+O(x^2)$ (Taylorreihe) verwendet: $k = \frac{\ln(1/2)}{\ln(1-N^{-1})} \approx \frac{\ln(1/2)}{-N^{-1}} = \frac{-\ln(2)}{-N^{-1}} = \ln(2)N$.

Mallory muss also bei einem 128-Bit Hashwert ca. $k = 2^{128}$ Änderungen durchprobieren, um mit 50-prozentiger Wahrscheinlichkeit einen Datensatz mit dem gewünschten Hashwert zu finden. Fassen wir zusammen:

Satz 4.23 (Brute-Force-Aufwand Urbildangriff / 2-tes Urbildangriff)
Bei einem **Angriff auf schwache Kollisionsresistenz (2-tes Urbild Angriff)** sucht ein Angreifer zu einem gegebenen Dokument x mit Hashwert h (der Länge n) ein zweites Dokument $x' \neq x$ mit demselben Hashwert. Der Brute-Force-Aufwand dafür ist etwa gleich

$$2^n,$$

d.h., er muss 2^n Dokumente zufällig probieren, um mit 50%-iger Wahrscheinlichkeit eine Kollision zu finden.

Der Brute-Force-Aufwand für einen **Urbildangriff** (d.h. der Angreifer sucht zu vorgegebenem Hashwert h ein Dokument x mit diesem Hashwert) ist ebenso groß.

Es gibt aber noch eine zweite Angriffsmöglichkeit, bei der der Aufwand um ein Vielfaches geringer ist:

Geburtstagsangriff (= Angriff auf starke Kollisionsresistenz): Angenommen, Mallory kann den Vertrag zur Unterfertigung mitbringen. Das könnte er wie folgt vorbereiten: Er beginnt mit zwei Verträgen, mit der echten und der falschen Kaufsumme, und führt an *beiden* so lange unauffällige Änderungen durch, bis der Hashwert von *irgendeinem* Vertrag x_1 mit der echten Kaufsumme mit dem Hashwert von *irgendeinem* Vertrag x_2 mit der falschen Kaufsumme übereinstimmt.

Betrachten wir nun wieder den Aufwand für einen solchen Angriff. Die Wahrscheinlichkeit für *irgendeine* Kollision (also irgendein Paar x_1, x_2 von Dateien mit demselben Hashwert h) unter k zufällig gewählten Dateien ist

$$1 - \frac{N(N-1)\cdots(N-k+1)}{N^k} = 1 - \frac{N!}{(N-k)!N^k}$$

(Gegenwahrscheinlichkeit zum Ereignis k verschiedene Hashwerte). Damit diese Wahrscheinlichkeit zumindest 50% wird, kann man zeigen, dass

$$k \approx \sqrt{2\ln(2)N} = \sqrt{2\ln(2)2^n} \approx 2^{n/2} = 2^{64}$$

sein muss (n ist die Länge des Hashwertes, hier haben wir im Beispiel $n = 128$ Bit angenommen; $N = 2^n$ bedeutet die Anzahl aller möglichen Hashwerte).

Das ist etwas schwieriger zu sehen. Wir beginnen wieder mit der Gleichung $1/2 = 1 - N!/((N-k)!N^k)$, also $N!/((N-k)!N^k) = 1/2$. Diese Gleichung lässt sich leider nicht nach k auflösen. Da wir aber gar nicht an der exakten Lösung, sondern nur an einem Näherungswert interessiert sind, können wir versuchen, die Gleichung zu nähern, um sie danach zu lösen. Dazu schreiben wir sie etwas um:

$$\frac{1}{2} = \prod_{j=0}^{k-1} (1 - \frac{j}{N})$$

Nun verwenden wir die Taylornäherung $\ln(1+x) = x + O(x^2)$ bei $x = 0$, um $(1 - \frac{j}{N}) = \mathrm{e}^{-j/N+O(k^2/N^2)}$ zu schreiben (hier haben wir gleich $O(j^2/N^2) = O(k^2/N^2)$ verwendet, da ja $j \leq k$). Eingesetzt und mit der Gauß'schen Summenformel $\sum_{j=1}^{k-1} j = \frac{k(k-1)}{2}$ vereinfacht erhalten wir

$$\frac{1}{2} = \exp\left(\frac{-k(k-1)}{2N} + O(\frac{k^3}{N^2})\right).$$

Nun lösen wir nach k auf und erhalten

$$k^2 + O(k) = 2\ln(2)N + O(\frac{k^3}{N}).$$

Diese Bedingung ist erfüllt, wenn wir $k \approx \sqrt{2\ln(2)N}$ wählen, da in diesem Fall die Fehlerterme auf beiden Seiten gleich $O(\sqrt{N})$ sind.

Die Rechnung würde natürlich einfacher, wenn man die Fehlerterme $O(\ldots)$ immer gleich weglässt. Wenn aber, so wie hier, zwei Größen k und N gegen Unendlich gehen, dann ist nicht von vornherein klar, welche Terme am Ende vernachlässigbar sind. Deswegen müssen die O-Terme bis zum Ende mitgeschrieben werden um sicherzustellen, dass kein wichtiger Term vernachlässigt wurde.

Der Aufwand für diesen Angriff ist also um ein Vielfaches geringer als beim 2-ten Urbildangriff zuvor.

Satz 4.24 (Brute-Force-Aufwand Geburtstagsangriff)
Bei einem **Angriff auf starke Kollisionsresistenz (Kollisionsangriff, Geburtstagsangriff)** sucht ein Angreifer irgendwelche Dokumente $x' \neq x$ mit demselben Hashwert h. Der Brute-Force-Aufwand dafür ist etwa gleich

$$2^{n/2},$$

(wobei $n \ldots$ Hashwertlänge), d.h., der Angreifer muss $2^{\frac{n}{2}}$ Dokumente zufällig probieren, um mit 50%-iger Wahrscheinlichkeit eine Kollision zu finden.

Dieser Angriff lässt sich verhindern, indem man jeden Vertrag leicht modifiziert (und somit einen allfälligen konstruierten Hashwert ändert), bevor man ihn signiert. In der Praxis wird das aber nicht immer möglich sein. Man muss daher die Länge n des Hashwertes so groß wählen, dass man auch gegen diesen Angriff sicher ist.

Erinnern Sie sich an das Geburtstagsparadoxon, das diesem Angriff den Namen gegeben hat: Nehmen wir an, Sie ordnen jeder Person in einem Raum ihren Geburtstag zu. Eine Anzahl von Personen wird also gleichmäßig auf 365 Plätze verteilt (wir nehmen an, dass jeder Geburtstag gleich wahrscheinlich ist). Irgendeine Kollision tritt auf, wenn *irgendwelche* zwei Personen am gleichen Tag Geburtstag haben. Die Wahrscheinlichkeit dafür ist bei 23 Personen $1 - \frac{365!}{342!365^{23}} = 0.507$, also über 50%! Wenn Sie also bei einer Party mit mindestens 23 Personen wetten, dass *irgendwelche* zwei Gäste am gleichen Tag Geburtstag haben, sind Ihre Chancen zu gewinnen größer als 50%! Wetten Sie darauf, dass jemand am *gleichen* Tag wie Sie Geburtstag hat, dann ist die Wahrscheinlichkeit bei 23 Gästen nur $1 - (1 - 365^{-1})^{22} = 0.058571$, also wohl kaum eine Wette wert.

MD5 gilt aufgrund von entdeckten Schwachstellen in der Merkle-Damgård-Konstruktion schon seit längerem als unsicher. Auf diesem Konstruktionsprinzip basieren auch der **SHA1** (Secure-Hash-Algorithmus), der Daten einen 160-Bit Hashwert zuordnet, und der **SHA2**, der für eine Familie von sechs Funktionen steht, mit Hashwerten von 224, 256, 384 oder 512 Bits. Für SHA1 hat man inzwischen eine Kollision gefunden [STE17].

Als Nachfolge-Algorithmus wird aktuell **SHA3** (SHA3-224, SHA3-256, SHA3-384 und SHA3-512) empfohlen, der seit 2012 nach einer internationalen Ausschreibung US-Standard ist ([NIST12], [FIPS180-4], [FIPS202]). Empfehlungen des BSI zur Verwendung von Hashfunktionen finden sich in [BSI].

SHA3 (vor der Standardisierung als **Keccak** bekannt) basiert auf einem anderen Prinzip, dass als **Schwamm-Konstruktion** [Ber+11] bezeichnet wird. Dabei wird die Nachricht x wie immer zunächst in Blöcke gleicher Länge zerlegt (und ggf. aufgefüllt). Dann wird der Status des Schwamms mit $s_0 = 0$ initialisiert und in zwei Teile $s_0 = (r_0, c_0)$ geteilt. In der Aufsaug-Phase wird nun der erste Teil mit der Nachricht verknüpft $r_j \oplus x_j$ und dann mit der Schwamm-Funktion verschleiert $(r_{j+1}, c_{j+1}) = f(r_j \oplus x_j, c_j)$ (hier wird wie üblich in mehreren Runden linear gemischt, permutiert und ein nichtlinearer Bestandteil verwendet). Ist die gesamte Nachricht aufgesaugt, kommt die Ausquetsch-Phase. Hier wird mehrfach r extrahiert, und dazwischen die Schwamm-Funktion angewendet, bis genügend Bits aus dem Schwamm gepresst wurden. Da im Prinzip beliebig viele Bits ausgepresst werden können, eignet sich die Konstruktion auch zur Erzeugung von Stromchiffren (in der Aufsaug-Phase wird dann der geheime Schlüssel aufgesaugt). In diesem Fall spricht man auch von einer **Extendable Output Function** (**XOF**), da man die Anzahl der extrahierten Bits frei wählen kann (Keccak in dieser Weise betrieben wird als **SHAKE** bezeichnet.)

Die Konstruktion ist in Abbildung 4.7 veranschaulicht. Die beiden Teile r und

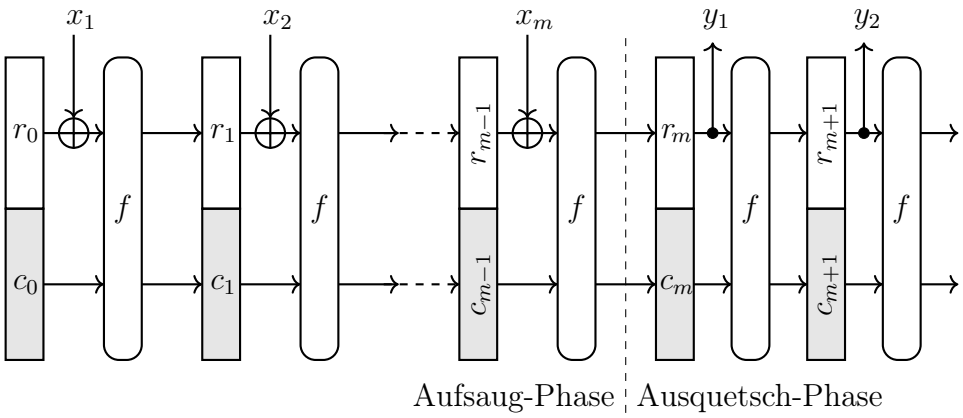

Abbildung 4.7: Schwamm-Konstruktion

c werden als (Bit-)Rate und Kapazität des Schwamms bezeichnet. Die Bitrate legt

fest, wie viele Bits pro Durchgang maximal ausgequetscht werden können. Die Kapazität legt fest, welcher Teil im Schwamm verbleibt und damit niemals offen gelegt wird. Ein Angreifer erhält somit keinerlei Informationen über diesen Teil des Status des Schwamms. Unter geeigneten Annahmen an f, kann man damit beweisen, dass das **Sicherheitsniveau** gegen Kollissions- bzw. Urbildangriffe mindestens die halbe Länge von c (in Bits) ist, falls auch (mindestens) so viele Bits extrahiert werden. Extrahiert man weniger Bits, so sinkt das Sicherheitsniveau natürlich auf die halbe Anzahl der extrahierten Bits. Das Sicherheitsniveau wird in der Regel als Zusatz angegeben, also SHA3-224 ist Keccak mit einer Kapazität von 448 Bits und 224 extrahierten Bits (die Summe aus Kapazität und Bitrate ist bei Keccak durch das verwendete f auf 1600 festgelegt; erhöht man die Kapazität, sinkt die Rate). Analog ist SHAKE128 Keccak mit einer Kapazität von 256, wobei eine variable Zahl d an Bits extrahiert wird. Das Sicherheitsniveau ist somit $\min(128, \frac{d}{2})$.

Die Schwamm-Konstruktion ist, im Gegensatz zur Merkle–Damgård-Konstruktion, auch resistent gegen Verlängerungsangriffe. D.h., es ist nicht möglich, ohne die gesamte Nachricht zu kennen, einen weiteren Block an die Nachricht zu hängen und einen dazu passenden Hashwert für die neue Gesamtnachricht zu generieren. Mehr dazu im nächsten Abschnitt.

HMACs

Hashfunktionen können auch zum Generieren von kryptographischen Prüfziffern, sogenannten **MACs** (Nachrichtenauthentifizierungscode, "Message authentication code"), verwendet werden. Solche Codes werden als **HMACs** bezeichnet.

Die Idee ist, dass der Hashwert der Nachricht verwendet werden kann, um zu überprüfen, ob die Nachricht verändert wurde. Natürlich müsste dann der Hashwert über einen Kanal übertragen werden, der von einem Angreifer nicht manipuliert werden kann. Deshalb müssen Alice und Bob ein geteiltes Geheimnis haben, dass sie in die Berechnung des Hashwertes einfließen lassen. Bleibt noch das Problem eines **Verlängerungsangriffs**, bei dem Mallory den Hashwert abfängt und nun weitere Nachrichtenblöcke an Bob sendet, die er am Ende mit einem gültigen Hashwert versieht.

Deshalb wird folgende Vorgehensweise zur Berechnung empfohlen [RFC2104]:

$$HMAC_k(x) = h\big((k \oplus o) \| h\big((k \oplus i) \| x\big)\big).$$

Dabei ist h die Hashfunktion, x die Nachricht, k der geheime Schlüssel (der die passende Blockgröße haben muss und ansonsten expandiert oder komprimiert werden muss) und i (*inner padding*) bzw. o (*outer padding*) sind fixe Bitmuster.

Das hier verwendete Konstruktionsprinzip $h(k\|h(k\|x))$ anstelle des einfacheren $h(k\|x)$ verhindert einen Verlängerungsangriff und wäre bei SHA3 nicht notwendig. Die andere einfache Möglichkeit $h(x\|k)$ hat das Problem, dass bei einem Hash, der auf der Merkle-Damgård-Konstruktion basiert, eine Kollision in der Nachricht $h(x) = h(x')$ automatisch auch eine Kollision der MACs bewirkt: $h(x\|k) = h(x) \oplus$

$f(h(x)\|k)) = h(x') \oplus f(h(x')\|k)) = h(x'\|k)$. Die beiden Bitmuster i und o spielen keine besondere Rolle, aber sie sollten nicht gleich sein. Im empfohlenen Standard wird $i = 1011100$ bzw. $o = 0110110$ so oft wiederholt bis die Blockgröße erreicht ist.

Bei der Verschlüsselung mit einer Blockchiffre wird ein MAC meist in der Form des **Galois Counter Mode** (**GCM**) verwendet. Hierbei handelt es sich um eine Variante des CTR, bei dem zusätzlich noch ein MAC über alle Blöcke gebildet wird. Dabei wird eine Blockgröße von 128 Bit gefordert und die verschlüsselten Blöcke y_j werden als Elemente in $GF(2^{128})$ (als irreduzibles Polynom wird $x^{128}+x^7+x^2+x+1$ verwendet) betrachtet und in jedem Schritt mit $h = E(0)$ multipliziert und per XOR mit dem vorherigen Ergebnis verknüpft. Ganz grob ergibt sich daraus folgende Erweiterung des Counter Modus:

$$y_j = z_j \oplus x_j, \quad a_j = (h * y_j) \oplus a_{j-1} \qquad z_j = E(t_j), \quad j = 1, \ldots, n,$$

wobei „$*$“ die Multiplikation in $GF(2^{128})$ bezeichnet und a_0 geeignet initialisiert werden muss. Der letzte Block a_n wird dann noch mit der Länge der Daten verknüpft und als Prüfziffer übermittelt. Beim Entschlüsselen wird auf die gleiche Weise a_j berechnet und am Ende mit dem übermittelten Wert verglichen. Für Details sei auf den entsprechenden Standard [Dwo07] verwiesen. Werden die verschlüsselten Daten nicht verwendet, reduziert sich GCM auf die Funktion als MAC und wird dann als **GMAC** (**Galois Message Authentication Code**) bezeichnet.

4.6 Passwörter

Auf Computern, an denen mehrere Personen arbeiten, oder in Netzwerken, ist es in der Regel notwendig, dass man sich authentifiziert. Die Authentifizierung geschieht dabei durch Angabe eines Benutzernamens (der öffentlich bekannt ist) und eines (geheimen) Passworts. Im einfachsten Fall hat ein Computer für jeden Benutzernamen das zugehörige Passwort gespeichert und kann dann durch einen einfachen Vergleich die Richtigkeit überprüfen. Durch entsprechende Berechtigungen muss sichergestellt werden, dass nur zuständige IT-Administrator:innen die Datei mit den Passwörtern lesen können.

In vielen Situationen ist es jedoch wünschenswert, dass auch IT-Administrator:innen die Passwörter nicht erfahren sollen (da zum Beispiel manche Personen wider allen Empfehlungen das gleiche Passwort auf verschiedenen Systemen verwenden) bzw. es muss für den Fall vorgesorgt werden, wenn die Datei mit den Passwörtern gestohlen wird. Zu diesem Zweck kann eine Einwegfunktion f zur Passwortverschlüsselung verwendet werden: Dabei berechnet man aus dem Passwort p einen geheimen Wert $c = f(p)$, der anstelle des Passwortes abgespeichert wird. Möchte sich nun eine Person authentifizieren, so wird aus dem eingetippten Passwort p' zunächst $c' = f(p')$ berechnet, und dann mit dem gespeicherten Wert c

verglichen. Wichtig dabei ist, dass die Funktion f so beschaffen ist, dass es keinen effizienten Angriff gibt um das Passwort p aus c zu berechnen (Einwegfunktion).

Zum Beispiel könnte man für die Funktion $f(p) = \mathrm{AES}_p(0)$ verwenden (Verschlüsselung der 0 mit dem Passwort p als Schlüssel). Da AES gegen Klartextangriffe sicher ist, ist gewährleistet, dass p nicht aus $\mathrm{AES}_p(0)$ berechnet werden kann! Die einzige Möglichkeit ist also, per Brute-Force alle möglichen Passwörter durchzuprobieren.

Und hier liegt aber die Schwachstelle des Verfahrens: der Mensch. Man kann sich nämlich in der Regel das Passwort selbst aussuchen. In der Praxis bedeutet das, dass die Passwörter oft aus einer Menge ausgewählt werden (leicht merkbare Passwörter), die um ein Vielfaches kleiner ist als die Menge aller möglichen Passwörter.

Ein Angreifer Mallory kann nun so vorgehen: er erzeugt ein *Wörterbuch* mit den gängigsten Passwörtern, berechnet $f(p)$ für alle diese Passwörter und vergleicht diese dann mit jenen in der Passwortdatei. Dabei knackt er zwar nicht alle Passwörter (eben nur die „schlechten“), aber die Chancen, zumindest in einigen Fällen fündig zu werden, stehen gut.

Um einen solchen **Wörterbuchangriff** zu erschweren, verwendet man einen sogenannten **Salt**-Wert. Das bedeutet, man wählt eine zufällige Zahl s und fügt sie (auf irgendeine Weise) zum Passwort p hinzu. Dann speichert man s und $c = f(p, s)$ gemeinsam ab. Nun müsste Mallory jedes Wort in seinem Wörterbuch mit allen möglichen Werten von s verschlüsseln und sein Aufwand vergrößert sich dementsprechend.

Analog funktioniert das CRYPT-Verfahren, das früher unter UNIX zur Passwortverschlüsselung verwendet wurde. Das Passwort darf dabei maximal 8 Zeichen lang sein, die mittels ASCII als $8 \cdot 7 = 56$-Bit Schlüssel für DES verwendet werden (wenn notwendig, wird mit Nullen aufgefüllt). Damit wird dann der Nullblock 25-Mal mit DES verschlüsselt (um den Rechenaufwand für einen Brute-Force-Angriff zu erhöhen). Zusätzlich wird ein 12-Bit Salt verwendet, mit dem der DES leicht modifiziert wird: Wenn das i-te Bit im Salt gleich eins ist, wird bei der Expansionspermutation das i-te mit dem $(i+24)$-ten Bit vertauscht (diese Modifikation soll verhindern, dass Standard-Hardwareimplementationen von DES für einen Brute-Force-Angriff verwendet werden können).

Als Nachfolger von CRYPT wurde ein auf MD5 basierender Algorithmus MD5-CRYPT verwendet, der die Passwortlänge nicht auf 8 Zeichen beschränkt, und Salt-Werte zwischen 12 und 48 Bit verwendet. Ein Wörterbuchangriff mit zuvor berechneten Werten ist bei genügend großem Salt zwar nicht mehr möglich, trotzdem können bei genügend großer Rechenleistung die gängigsten Passwörter durchprobiert werden. Bei schlecht gewählten Passwörtern kann auch der beste Algorithmus nicht helfen.

In diesem Zusammenhang sei noch erwähnt, dass ein vier-stelliger PIN in diesem Sinn kein sicheres Passwort ist (da zu kurz) und auch durch den besten Algorithmus nicht geschützt werden kann. Deshalb wird in modernen Handys aus dem PIN in einem eigenen Hardwaremodul ein

sicherer Schlüssel generiert. Der Schutz gegen Brute-Force-Angriffe wird dem Hardwaremodul übertragen, das die Anzahl der Versuche pro Zeit begrenzen muss (siehe z.B. https://blog.cryptographyengineering.com/2014/10/04/why-cant-apple-decrypt-your-iphone/).

Ein zusätzlicher Schutz gegen Brute-Force-Angriffe ist es, den *Aufwand* für eine einzelne Passwortverschlüsselung zu erhöhen. Der auf BLOWFISH basierende BCRYPT-Algorithmus enthält daher zusätzlich einen **Kostenparameter**, über den der Aufwand eingestellt werden kann. Natürlich kann man den Aufwand nicht beliebig hoch wählen, denn es muss ja das Einloggen in endlicher Zeit möglich sein.

Aktuell wird die **Password-Based Key Derivation Function** (**PBKDF**) empfohlen, die in [RFC8018] spezifiziert ist. Dabei wird aus dem Passwort p, dem Salt s und einem Kostenparameter c mithilfe einer Hashfunktion H der abgeleitete Schlüssel k (*derived key*) berechnet. In der einfacheren Version 1 geschieht das indem die Hashfunktion c Mal auf $p\|s$ angewandt wird:

$$k = \underbrace{(H \circ \cdots \circ H)}_{c \text{ mal}}(p\|s).$$

Der Wert k kann dann als verschlüsseltes Passwort abgespeichert werden, oder (daher der Name) als Schlüssel für eine Blockchiffre verwendet werden (in letzterem Fall muss er ggf. noch auf die passende Länge gekürzt werden). In Version 2 wird eine HMAC-Funktion mit dem Passwort als Schlüssel verwendet und die Iterierten Anwendungen werden alle zusammen mit XOR verknüpft. Ausserdem ist vorgesehen, dass man auch längere k Werte erzeugen kann indem das Verfahren j Mal mit Startwert $s\|1, \ldots, s\|j$ durchführt und die Ergebnisse zu einem Wert der gewünschten Länge zusammensetzt.

4.7 Kontrollfragen

Fragen zu Abschnitt 4.1: Data Encryption Standard (DES)

Erklären Sie folgende Begriffe und überprüfen Sie Ihre Antwort mit dem Skriptum: Diffusion, Konfusion, DES, Blockchiffre, Produktchiffre, Feistel-Box, Triple DES.

1. Nennen Sie zwei Schwachstellen der Vigenère-Chiffre.

   (Lösung zu Kontrollfrage 1)

2. Ist das Vertauschen beider Bits in einem Zwei-Bit-Block eine lineare oder eine nichtlineare Funktion?

   (Lösung zu Kontrollfrage 2)

3. Warum verwendet man zur Verschlüsselung nicht einfach nur eine „wilde“ Permutation der einzelnen Bits (innerhalb des Blockes)?

   (Lösung zu Kontrollfrage 3)

4. Wie viele Blöcke der Länge 8 gibt es über dem Alphabet $A = \{0, 1\}$?

   (Lösung zu Kontrollfrage 4)

5. Wie lang sind die Blöcke, auf denen DES operiert? Wie lang ist der Schlüssel von DES?

   (Lösung zu Kontrollfrage 5)

6. Richtig oder falsch? Bei einem Brute-Force-Angriff auf DES müssen im Durchschnitt etwa 2^{55} Schlüssel getestet werden.

   (Lösung zu Kontrollfrage 6)

7. Richtig oder falsch? DES ist eine monoalphabetische Chiffre bei einem Alphabet von 2^{64} Buchstaben.

   (Lösung zu Kontrollfrage 7)

8. Warum ist eine statistische Kryptoanalyse von monoalphabetischen Verfahren über einem Alphabet mit 2^{64} Buchstaben praktisch nicht möglich?

   (Lösung zu Kontrollfrage 8)

9. Eine Feistel-Funktion ist invertierbar, egal, wie der nichtlineare Inhalt f aussieht.

   (Lösung zu Kontrollfrage 9)

10. Was ist die Bedeutung einer Feistel-Box F_k?

    (Lösung zu Kontrollfrage 10)

11. Warum muss nach einer Feistel-Box jeweils die linke und die rechte Hälfte der Ausgangsbits vertauscht werden? Wie viele Feistel-Boxen werden beim DES durchlaufen?

    (Lösung zu Kontrollfrage 11)

12. Richtig oder falsch? Der nichtlineare Bestandteil f des DES in der Feistel-Box ist die Bildung des multiplikativen Inversen in $GF(2^8)$.

    (Lösung zu Kontrollfrage 12)

13. Wozu dienen die S-Boxen beim DES?

    (Lösung zu Kontrollfrage 13)

14. Wo geht der geheime Schlüssel beim DES ein?

    (Lösung zu Kontrollfrage 14)

15. Was ist ein Lawineneffekt? Wodurch kommt er beim DES zustande?

    (Lösung zu Kontrollfrage 15)

16. Was passiert bei einem Feistel-Netzwerk, wenn man in jeder Runde den gleichen Schlüssel verwendet? Warum ist das keine gute Idee?

    (Lösung zu Kontrollfrage 16)

17. Richtig oder falsch? Die Inverse der Permutation | 3 | 5 | 2 | 1 | 4 | (analog zur DES-Dokumentation FIPS46-3 in Tabellenform geschrieben) ist | 4 | 3 | 1 | 5 | 2 |.

    (Lösung zu Kontrollfrage 17)

18. Wie sicher ist der DES?

    (Lösung zu Kontrollfrage 18)

19. Warum wird nicht 2DES (Zweifachverschlüsselung mit DES) verwendet?

    (Lösung zu Kontrollfrage 19)

20. Was ist Triple DES? Wie lang ist der Schlüssel von Triple DES effektiv?

    (Lösung zu Kontrollfrage 20)

Fragen zu Abschnitt 4.2: Kryptoanalyse moderner Blockchiffren

Erklären Sie folgende Begriffe und überprüfen Sie Ihre Antwort mit dem Skriptum: Abhängigkeitsmatrix, lineare Kryptoanalyse, differentielle Kryptoanalyse.

1. Würde die differentielle Kryptoanalyse auch noch funktionieren, wenn der Schlüssel statt mit XOR mit AND verknüpft würde? Was ist der Vorteil von XOR im Vergleich zu AND?

   (Lösung zu Kontrollfrage 1)

2. Ist eine S-Box vor der ersten Schlüsseladdition sinnvoll?

   (Lösung zu Kontrollfrage 2)

3. Was versteht man unter Key Whitening? Was ist der Zweck?

   (Lösung zu Kontrollfrage 3)

Fragen zu Abschnitt 4.3: Der Advanced Encryption Standard (AES)

Erklären Sie folgende Begriffe und überprüfen Sie Ihre Antwort mit dem Skriptum: NIST, AES.

1. Welche Algorithmen kamen bei der Suche nach dem AES in die engere Wahl?

   (Lösung zu Kontrollfrage 1)

2. Richtig oder falsch: Rijndael wurde ausgesucht, weil sich herausgestellt hat, dass er am sichersten ist.

   (Lösung zu Kontrollfrage 2)

3. Welche Blocklängen und Schlüssellängen stehen bei AES zur Verfügung?

   (Lösung zu Kontrollfrage 3)

4. Richtig oder falsch? Feistel-Boxen werden sowohl beim AES als auch beim DES verwendet.

   (Lösung zu Kontrollfrage 4)

5. Welche Form hat eine lineare Transformation in einer Dimension?

   (Lösung zu Kontrollfrage 5)

6. Welche Form hat eine affine Transformation in einer Dimension?

   (Lösung zu Kontrollfrage 6)

7. Was ist die nichtlineare Transformation bei Rijndael?

   (Lösung zu Kontrollfrage 7)

8. Richtig oder falsch? AES ist perfekt sicher.

   (Lösung zu Kontrollfrage 8)

9. Richtig oder falsch? Bei AES wird in der AddRoundKey-Schicht der Zustand mit dem jeweiligen Rundenschlüssel durch eine bitweise XOR-Operation verknüpft.

   (Lösung zu Kontrollfrage 9)

10. Richtig oder falsch? In $GF(2^8)$ mit Modul $m(x) = x^8 + x^4 + x^3 + x + 1$ ist der Kehrwert von x gleich x^7.

    (Lösung zu Kontrollfrage 10)

11. Richtig oder falsch? Die SubBytes-Schicht enthält den nichtlinearen Anteil von AES.

    (Lösung zu Kontrollfrage 11)

Fragen zu Abschnitt 4.4: Betriebsarten

Erklären Sie folgende Begriffe und überprüfen Sie Ihre Antwort mit dem Skriptum: Betriebsmodi, Padding, Initialisierungsvektor.

1. Welche Betriebsmodi für Blockchiffren gibt es, die dem Zweck der Verschlüsselung dienen?

   (Lösung zu Kontrollfrage 1)

2. Kann eine Blockchiffre auch als Stromchiffre betrieben werden?

   (Lösung zu Kontrollfrage 2)

Fragen zu Abschnitt 4.5: Kryptographische Hashfunktionen

Erklären Sie folgende Begriffe und überprüfen Sie Ihre Antwort mit dem Skriptum: Kryptographische Hashfunktion, schwach kollisionsresistent, stark kollisionsresistent, Kollision, Einweg-Hashfunktion, Geburtstagsangriff, HMAC.

1. Was ist eine Hashfunktion?

   (Lösung zu Kontrollfrage 1)

2. Was versteht man im Zusammenhang mit einer Hashfunktion unter einer Kollision?

   (Lösung zu Kontrollfrage 2)

3. Warum muss eine Hashfunktion zur Passwortverschlüsselung schwach kollisionsresistent sein?

   (Lösung zu Kontrollfrage 3)

4. Was ist eine Einwegfunktion?

   (Lösung zu Kontrollfrage 4)

5. Richtig oder falsch: Der Vorteil von SHA3 gegenüber MD5 ist, dass zwei verschiedene Dateien auch immer verschiedene Fingerabdrücke haben.

   (Lösung zu Kontrollfrage 5)

6. Was ist der Geburtstagsangriff? (Mehrere richtige Antworten möglich):
   a) Man versucht, zwei verschiedene Dokumente (z. B. echter und gefälschter Vertragstext) zu finden, die den gleichen Hashwert (*egal, welchen*) besitzen.
   b) Man versucht, zwei verschiedene Dokumente (z. B. echter und gefälschter Vertragstext) zu finden, die einen *bestimmten* Hashwert besitzen.
   c) Man versucht, den Hashwert eines echten Dokumentes durch unwesentliche Änderungen so zu ändern, dass sich derselbe Hashwert wie beim gefälschten Dokument ergibt.
   d) Man versucht, ein Dokument zu erzeugen, dessen Hashwert gleich dem Geburtstag des Verfassers ist.

   (Lösung zu Kontrollfrage 6)

7. Wie viele zufällig gewählte Dateien müssen von Mallory probiert werden, damit er (mit zumindest 50%iger Wahrscheinlichkeit) eine Datei findet, die einen bestimmten (vorgegebenen) Hashwert hat? (Hashwert hat Länge n.) Wie nennt man diesen Angriff?

   (Lösung zu Kontrollfrage 7)

8. Wie viele zufällig gewählte Dateien müssen von Mallory probiert werden, damit er (mit zumindest 50%iger Wahrscheinlichkeit) zwei (oder mehr) Dateien findet, die denselben Hashwert haben? (Hashwert hat Länge n.) Wie nennt man diesen Angriff?

   (Lösung zu Kontrollfrage 8)

9. Würden Sie bei der Konstruktion einer Hashfunktion die Nachrichtenblöcke x_j durch Abschneiden auf eine kleinere Länge bringen (damit man sie z.B. in einer Verschlüsselungsfunktion mit kleinere Blocklänge verwenden kann)?

   (Lösung zu Kontrollfrage 9)

Fragen zu Abschnitt 4.6: Passwörter

Erklären Sie folgende Begriffe und überprüfen Sie Ihre Antwort mit dem Skriptum: Wörterbuchangriff, Salt.

1. Was ist ein Wörterbuchangriff?

   (Lösung zu Kontrollfrage 1)

2. Was bedeutet *Salt* im Zusammenhang mit Passwortverschlüsselung? Wozu wird es verwendet?

   (Lösung zu Kontrollfrage 2)

3. Wenn sich Ihr Chef weigert, ein Passwort mit mehr als drei Buchstaben zu wählen: wie groß muss der Salt-Wert mindestens sein, damit Ihre Passwortdatei einem Brute-Force-Angriff standhält?

   (Lösung zu Kontrollfrage 3)

Lösungen zu den Kontrollfragen

Lösungen zu Abschnitt 4.1

1. keine Diffusion, keine nichtlinearen Bestandteile

2. Linear, denn die Verschlüsselungsvorschrift lautet:

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}.$$

3. Weil Permutationen der Bits lineare Funktionen sind. Lineare Gleichungssysteme können effektiv gelöst werden und ein Klartextangriff ist leicht möglich.

4. 2^8

5. DES verschlüsselt 64-Bit Blöcke in 64-Bit Blöcke. Der Schlüssel ist effektiv 56 Bit lang, zusätzlich enthält er 8 Prüfbits.

6. Richtig. Effektive Schlüssellänge ist 56 Bit, somit ergibt sich für den durchschnittlichen Aufwand (Durchsuchen des halben Schlüsselraums) 2^{55} Schlüssel.

7. Richtig.

8. Weil dazu ein viel zu langer Geheimtext zur Analyse vorliegen müsste, damit die einzelnen Buchstaben mit repräsentativen Häufigkeiten vorkommen.

9. Richtig.

10. Sie sichert aufgrund ihrer speziellen Konstruktion, dass die Verschlüsselungsvorschrift invertierbar ist. Es ist insbesondere immer $F_k^{-1} = F_k$.

11. Weil die Feistel-Box die rechte Hälfte der Eingangsbits unverändert als rechte Hälfte der Ausgangsbits wieder ausgibt. Ohne Vertauschung würde daher ein Teil des Klartextes nicht verschlüsselt werden (abgesehen von der Eingangspermutation). Beim DES werden 16 Feistel-Boxen (16 „Runden“) durchlaufen.

12. Falsch. Die Bildung des multiplikativen Inversen in $GF(2^8)$ ist der nichtlineare Bestandteil beim AES.

13. Sie sind der nichtlineare Anteil und damit ein entscheidender Faktor für die Sicherheit.

14. Aus dem geheimen Schlüssel werden 16 Teilschlüssel erzeugt. Diese gehen in den 16 Runden in die nichtlineare Funktion f innerhalb der Feistel-Boxen ein.

15. Lawineneffekt bedeutet, dass die Änderung eines einzigen Klartextbits jedes Geheimtextbit mit der Wahrscheinlichkeit $\frac{1}{2}$ ändert. Er kommt beim DES durch die Expansionspermutation E am Eingang und durch die P-Permutation am Ausgang der f-Funktion in der Feistel-Box zustande gemeinsam mit der Tatsache, dass insgesamt 16 Runden durchlaufen werden.

16. Jeweils 4 Runden heben sich auf.

17. Richtig.

18. Der einzige mögliche Angriff ist Brute-Force. Aufgrund der mittlerweile zu kleinen Schlüssellänge ist DES heute nicht mehr resistent gegen einen Brute-Force-Angriff.

19. Weil ein Angreifer bei einem Meet in the Middle Angriff nur den doppelten Aufwand hätte wie bei einem Brute-Force-Angriff auf DES. Der effektive Schlüssel von 2DES ist daher nicht doppelt so lang wie von DES sondern nur um ein Bit länger.

20. Die dreifache Anwendung von DES. Die effektive Schlüssellänge von Triple DES (mit drei unabhängigen Schlüsseln) ist aufgrund der Möglichkeit eines Meet in the Middle Angriffs $2 \cdot 56 = 112$ Bit.

Lösungen zu Abschnitt 4.2

1. Nein, denn AND ist nichtlinear und der Schlüssel würde sich beim Bilden der Differenz nicht herauskürzen. Bei AND ist das Ergebnis in drei von vier Fällen gleich 0 und das würde den Einfluß des Schlüssels auf den Geheimtext reduzieren.

2. Nein, denn sie könne bei einem Angriff leicht herausgerechnet werden.

3. Eine Schlüsseladdition am Beginn und am Ende des Algorithmus. Ein Angreifer kann die ersten/letzten Schritte des Algorithmus bei einem Angriff nicht herausrechnen.

Lösungen zu Abschnitt 4.3

1. MARS, RC6, Rijndael, Serpent und Twofish.

2. Falsch. Alle fünf Algorithmen gelten als gleich sicher.

3. Blocklänge und Schlüssellänge können unabhängig voneinander die Werte 128, 192 und 256 Bit annehmen.

4. Falsch; DES ist ein Feistelnetzwerk, AES aber nicht.

5. $y = f(x) = ax$ (a entspricht der Matrix); also z.B. $y = 2x$; der Graph von f ist eine Gerade durch den Ursprung.

6. $y = f(x) = ax + d$ (a entspricht der Matrix, d ist der feste „Vektor“ der Dimension 1); also zum Beispiel $y = 2x + 1$. Der Graph von f ist nun eine Gerade, die für $d \neq 0$ nicht durch den Ursprung geht.

7. Die Berechnung des multiplikativen Inversen in $GF(2^8)$.

8. Falsch. AES ist *praktisch* sicher.

9. Richtig.

10. Falsch. $x \cdot x^7 = x^8 \neq 1$ modulo $m(x) = x^8 + x^4 + x^3 + x + 1$ (denn mit dem „Trick“ sieht man sofort $x^8 = x^4 + x^3 + x + 1$).

11. Richtig.

Lösungen zu Abschnitt 4.4

1. ECB, CBC, CFB, OFB, CTR

2. Ja, mit CFB, OFB oder CTR.

Lösungen zu Abschnitt 4.5

1. Eine Funktion h, die Daten beliebiger Länge auf Daten mit fester Länge n abbildet. Man nennt $h(x)$ den Hashwert von x.

2. Das ist ein Paar von verschiedenen Datensätzen x und x', die denselben Hashwert besitzen. Bei einer Hashfunktion gibt es *immer* Kollisionen, da es unendlich viele Datensätze beliebiger Länge, aber nur endlich viele der Länge n gibt.

3. Da man sonst ein (im Allgemeinen anderes) Passwort mit dem gleichen Hashwert finden kann, dass also ebenfalls als gültig akzeptiert wird.

4. Eine Funktion h heißt Einwegfunktion, wenn es praktisch nicht möglich ist, zu vorgegebenem Funktionswert s ein x zu finden mit $h(x) = s$.

5. Falsch. Kollisionen sind unvermeidbar. Es ist nur aufgrund des längeren Hashwertes bei SHA3 derzeit praktisch unmöglich, eine Kollision zu finden.

6. a) richtig b) falsch c) falsch d) falsch

7. 2^n Dateien; Angriff auf schwache Kollisionsresistenz (2-tes Urbildangriff).

8. $2^{n/2}$ Dateien; Geburtstagsangriff.

9. Nein, dann durch Ändern des abgeschnittenen Teils (der dann nicht in die Berechnung eingeht) kann man in diesem Fall leicht Nachrichten mit gleichen Hashwerten erzeugen.

Lösungen zu Abschnitt 4.6

1. Ein Angriff, bei dem die wahrscheinlichsten Passwörter (inklusive aller möglichen Salt-Werte) verschlüsselt und abgespeichert werden. Danach braucht nur noch mit der Passwortdatei verglichen werden.

2. Eine Zusatzzahl, um die das Passwort erweitert wird, um Wörterbuchangriffe zu erschweren. (Außerdem ist dann aus der Passwortdatei nicht mehr ersichtlich, wenn zwei Personen das gleiche Passwort haben).

3. Egal, wie lange der Salt-Wert ist oder wie sicher Ihr Algorithmus ist: es ist immer möglich alle Kombinationen aus drei Buchstaben durchzuprobieren.

4.8 Übungen

Aufwärmübungen

1. **Eingangspermutation bei DES**: Was ist bei DES das Ergebnis der Eingangspermutation $IP(x)$, wenn $x = 0 \dots 010 \dots 0$ (alle 64 Bits sind 0 bis auf das Bit Nummer 26)? (Verwenden Sie FIPS46-3.)

2. **Expansionspermutation bei DES**: Was ist das Ergebnis der Expansionspermutation $E(r)$ (zu Beginn der Feistel-Box), wenn $r = 00001 \dots 0$ (alle 32 Bits sind 0 bis auf das Bit Nummer 5)? (Verwenden Sie FIPS46-3.)

3. **S-Box bei DES**: Geben Sie den Ausgang der S_1-Box mit dem Input 110010 an. In welchem Teil von DES befindet sich die S_1-Box? In welcher Runde wird sie verwendet? (Verwenden Sie FIPS46-3.)

4. Berechnen Sie die ersten 8 Bits des Ausgangs der S-Box von DES, wenn in die S-Box die 48-Bitfolge $00000101 \dots 0$ (alle Bits sind 0 bis auf die Bits Nummer 6 und 8) eingegeben wird. (Verwenden Sie FIPS46-3.)

5. **Ergebnis einer Runde von DES**: Der Klartextblock $x = 0 \dots 0$ (alle 64 Bits sind 0) wird mit DES verschlüsselt. Berechnen Sie das Ergebnis der ersten Runde, also
$$y = (\Theta \circ F_{k_1} \circ IP)(x),$$
wenn dabei der Rundenschlüssel $k_1 = 0 \dots 0$ (alle Bit gleich 0) verwendet wird. Geben Sie die 64 Bit des Ergebnisses in Blöcken zu 4 Bit zusammengefasst (und diese hexadezimal geschrieben) an. (Verwenden Sie FIPS46-3.)

6. Berechnen Sie die folgenden Produkte bzw. Summen von Bytes (diese sind nun hexadezimal dargestellt) i indem Sie die Bytes mit Polynomen aus

$\mathbb{Z}_2[x]_{x^8+x^4+x^3+x+1}$ identifizieren:
a) $03 \cdot 03$ b) $04 + 02$ c) $03 \cdot 04$ d) $8C \cdot 02$

7. **SubBytes-Transformation von AES**: Was ist das Ergebnis (der Zustand) nach der SubBytes-Schicht von AES, wenn die Zustandsmatrix

$$\begin{pmatrix} C3 & 00 & 00 & 00 \\ C3 & 00 & 00 & 00 \\ C3 & 00 & 00 & 00 \\ C3 & 00 & 00 & 00 \end{pmatrix}.$$

(alle Elemente der ersten Spalte sind gleich dem Byte $C3$ in Hex-Darstellung, die restlichen sind gleich 00) in SubBytes eingegeben wird?

Ermitteln Sie dieses Ergebnis mithilfe der Tabelle in der AES Dokumentation FIPS197.

8. **SubBytes-Transformation von AES**: Gegeben ist der Zustand

$$\begin{pmatrix} 00 & 00 & 00 & 01 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \end{pmatrix}.$$

Berechnen Sie das Ergebnis der SubBytes-Transformation und kontrollieren Sie am Ende mithilfe der Tabelle (Figure 7) in der AES Dokumentation FIPS197.

9. **MixColumns-Transformation von AES**: Gegeben ist der Zustand

$$\begin{pmatrix} 00 & 00 & 00 & 00 \\ 03 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 02 & 00 & 00 & 00 \end{pmatrix}.$$

Berechnen Sie das Ergebnis, wenn dieser Zustand in die MixColumns-Schicht eingegeben wird. (Verwenden Sie die AES Dokumentation FIPS197.)

10. **ShiftRows-Transformation von AES**: Gegeben ist der Zustand

$$\begin{pmatrix} A1 & 2F & 01 & 05 \\ A1 & 2F & 01 & 05 \\ A1 & 2F & 01 & 05 \\ A1 & 2F & 01 & 05 \end{pmatrix}.$$

Berechnen Sie das Ergebnis dieses Zustandes nach der ShiftRows-Transformation. (Verwenden Sie FIPS197.)

11. Verwenden Sie die Beziehung

$$x_1 + x_2 + x_4 + y_1 = 0$$

für den ersten Teil der ersten S-Box $S^1 : x_1x_2x_3x_4 \mapsto y_1y_2$ beim S-DES um eine affine Beziehung für den S-DES zu finden.

12. **Betriebmodi einer Blockchiffre**: Gegeben ist die (einfache) Blockchiffre E, die einen 4-Bit-Block $b_1b_2b_3b_4$ nach folgender Vorschrift verschlüsselt:

$$E(b_1b_2b_3b_4) = b_2b_3b_4b_1.$$

Weiters ist der Klartext 1011 0001 1011 gegeben, also drei 4-Bit Blöcke. Verschlüsseln Sie den Klartext
a) im ECB-Mode
b) im CBC-Mode, wenn der Initialisierungsvektor $IV = 1100$ ist.
Entschlüsseln Sie jeweils wieder. Handelt es sich um eine Betriebsart als Block- oder als Stromchiffre? Wie wirkt sich ein falsch übertragenes Bit im ersten Geheimtextblock bei der Entschlüsselung auf die Klartextblöcke aus?

Weiterführende Aufgaben

1. **Eingangspermutation bei DES**: Was ist bei DES das Ergebnis der Eingangspermutation $IP(x)$, wenn $x = 0 \ldots 010 \ldots 0$ (alle 64 Bits sind 0 bis auf das Bit Nummer 33)? (Verwenden Sie die DES-Dokumentation FIPS46-3.)

2. **Expansionspermutation bei DES**: Was ist das Ergebnis der Expansionspermutation $E(r)$ (zu Beginn der f-Funktion), wenn $r = 00010 \ldots 0$ (alle 32 Bits sind 0 bis auf das Bit Nummer 4)? (Verwenden Sie die DES-Dokumentation FIPS46-3.)

3. **S-Box bei DES**: Berechnen Sie den Ausgang der S-Box von DES, wenn in die S-Box die 48-Bitfolge $0000101 \ldots 0$ (alle Bits sind 0 bis auf die Bits Nummer 5 und 7) eingegeben wird. (Geben Sie die gesuchten 32 Bit des Ausgangs aus Gründen der Übersichtlichkeit in Blöcken von 4 Bit zusammengefasst an, und stellen Sie jeden hexadezimal dar.)
Hinweis: Die eingegebenen 48 Bits sind in Blöcke von je 6 Bits zu zerlegen, die in die Boxen $S_1, \ldots S_8$ eingegeben werden.
(Verwenden Sie FIPS46-3.)

4. **Ergebnis einer Runde von DES**: Der Klartextblock $x = 0 \ldots 010 \ldots 0$ (alle 64 Bits sind 0 bis auf das Bit Nr. 33) wird mit DES verschlüsselt. Berechnen Sie das Ergebnis der ersten Runde, also

$$y = (\Theta \circ F_{k_1} \circ IP)(x),$$

wenn dabei der Rundenschlüssel $k_1 = 0 \ldots 0$ (alle Bits gleich 0) verwendet wird. Geben Sie die 64 Bit des Ergebnisses in Blöcken zu 4 Bit zusammengefasst (und diese hexadezimal geschrieben) an. (Verwenden Sie die DES-Dokumentation FIPS46-3.)
Hinweis: Sie können teilweise die Ergebnisse der vorangehenen Übungen 1, 2 und 3 verwenden :-)

5. Beweisen Sie die **Komplementeigenschaft** eines Feistelnetzwerks E_k:

$$\overline{E_k(x)} = E_{\overline{k}}(\overline{x}),$$

wobei $\overline{x}$ das bitweise Komplement von x bezeichnet ($\overline{010} = 101$, etc.). Die Schlüssellänge von Feistel-Netzwerken (DES, S-DES) ist also effektiv um 1 reduziert. (Hinweis: Was passiert beim bitweisen Komplement einer XOR-Verknüpfung?)

6. Ein Schlüssel bei dem (nach entfernen der Kontrollbits) die linke und rechte Hälfte entweder nur aus Nullen oder nur aus Einsen besteht wird als **schwacher DES-Schlüssel** bezeichnet. Es gibt also vier schwache Schlüssel. Was passiert bei Verschlüsselung mit einem solchen Schlüssel?

7. Wie viel Speicher wird für einen Meet-in-the-Middle-Angriff auf 3DES benötigt?

8. Zeigen Sie, dass die Permutation

$$P(x_1x_2x_3x_4x_5x_6) = x_3x_4x_5x_6x_1x_2$$

eine *lineare Abbildung* von $\mathbb{Z}_2^6 \to \mathbb{Z}_2^6$ ist. ist. Für den Beweis gibt es zwei Alternativen (versuchen Sie zumindest eine davon):

a) Zeige, dass gilt:

- $P(\mathbf{a} + \mathbf{b}) = P(\mathbf{a}) + P(\mathbf{b})$ für beliebige Vektoren $\mathbf{a}$ und $\mathbf{b}$ aus $\mathbb{Z}_2^6$
- $P(k\mathbf{a}) = kP(\mathbf{a})$ für beliebige Vektoren $\mathbf{a}$ aus $\mathbb{Z}_2^6$ und beliebige $k \in \mathbb{Z}_2$.

b) Zeige, dass sich die Funktionsvorschrift mithilfe einer Matrix schreiben lässt, dass es also eine Matrix A gibt mit:

$$P\left(\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix}\right) = A \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix}.$$

9. Schreiben Sie die klassischen Logikfunktionen $NOT(x_1)$, $OR(x_1, x_2)$, $AND(x_1, x_2)$, $XOR(x_1, x_2)$ als Polynome in $\mathbb{Z}_2$. Welche dieser Funktionen sind linear?

10. Betrachten Sie die S-Box $\mathbb{Z}_2^3 \to \mathbb{Z}_2^3$ die durch folgende Vorschrift gegeben ist:

$$\begin{aligned} y_1 &= x_1x_2 + x_1x_3 + x_2x_3 + x_2 + x_3 + 1 \\ y_2 &= x_1x_2 + x_1x_3 + x_2x_3 + x_1 + x_3 + 1 \\ y_3 &= x_1x_2 + x_1x_3 + x_2x_3 + x_1 + x_2 + 1 \end{aligned}$$

Diese S-Box ist offensichtlich nichtlinear und jedes Eingangsbit x_j beeinflusst jedes Ausgangsbit x_k. Würden Sie diese S-Box in einer Verschlüsselungsvorschrift verwenden?

11. Berechnen Sie die Abhängigkeitsmatrix von

$$\begin{aligned} y_1 &= x_1x_2 + x_1x_3 + x_2x_3 + x_2 + x_3 + 1 \\ y_2 &= x_1x_2 + x_1x_3 + x_2x_3 + x_1 + x_3 + 1 \\ y_3 &= x_1x_2 + x_1x_3 + x_2x_3 + x_1 + x_2 + 1 \end{aligned}$$

12. Berechnen Sie die Biastabelle der nichtlinearen Funktion

$$\begin{aligned} y_1 &= x_1 + x_2 + x_1x_2 \\ y_2 &= x_2 + x_1x_2 \end{aligned}$$

Gibt es lineare Beziehungen zwischen den Variablen?

13. Gibt es eine nichtaffine invertierbare S-Box $\mathbb{Z}_2^2 \to \mathbb{Z}_2^2$?
Hinweis: Wie viele invertierbare affine Funktionen gibt es?

14. Betrachten Sie folgendes einfache Verfahren mit einer großen S-Box und einer Runde:

$$E_k(x) = (X_k \circ S)(x)$$

Hier ist $X_k(x) = x \oplus k$ die Schlüsseladdition (per XOR) und S die (invertierbare) S-Box.

Ist dieses Verfahren sicher, wenn die Block-/Schlüssellänge genügend groß und die (nichtlineare) S-Box gut gewählt ist? Wenn nicht, wie könnte es angegriffen werden? Wäre es besser die Operationen zu vertauschen (also $E_k(x) = (S \circ X_k)(x)$) oder überhaupt gleich zwei S-Boxen zu verwenden (also $E_k(x) = (S_2 \circ X_k \circ S_1)(x)$)? Wie sieht es aus, wenn Key-Whitening verwendet wird (also $E_k(x) = (X_k \circ S \circ X_k)(x)$)?

15. Zeigen Sie, dass die S_1-Box von DES (in der f-Funktion) *keine* lineare Abbildung von $\mathbb{Z}_2^6 \to \mathbb{Z}_2^4$ ist, dass also nicht für alle 6-Bit Folgen $\mathbf{a} = a_1a_2a_3a_4a_5a_6$ und $\mathbf{b} = b_1b_2b_3b_4b_5b_6$ und alle $k \in \mathbb{Z}_2$ gilt:

$$\begin{aligned} S_1(\mathbf{a}+\mathbf{b}) &= S_1(\mathbf{a}) + S_1(\mathbf{b}) \\ S_1(k\mathbf{a}) &= kS_1(\mathbf{a}). \end{aligned}$$

Hinweis: Verwenden Sie die DES-Dokumentation FIPS46-3.

16. Falls bei S-DES für einen Schlüssel k die beiden Teilschlüssel gleich sind, also $k^1 = k^2$, so ist

$$SDES_k \circ SDES_k(x) = x.$$

Finden Sie alle solchen Schlüssel für den S-DES.

17. Verwenden Sie die Beziehung

$$x_1 + x_3 + x_4 + y_2 = 0,$$

für den zweiten Teil der S-Box $S^2 : x_1x_2x_3x_4 \mapsto y_1y_2$ beim S-DES um eine affine Beziehung für den S-DES zu finden.

18. **SubBytes-Transformation bei AES:** Gegeben ist der Zustand

$$\begin{pmatrix} 00 & 00 & 00 & C3 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \end{pmatrix}.$$

Berechnen Sie das Ergebnis der SubBytes-Transformation und kontrollieren Sie am Ende mithilfe der Tabelle (Figure 7) in der AES Dokumentation FIPS197. Hinweis: $(x^7 + x^6 + x + 1)(x^7 + x^5 + x + 1) = 1$ modulo $m(x) = x^8 + x^4 + x^3 + x + 1$.

19. **SubBytes-Transformation bei AES:** Was ist das Ergebnis (der Zustand) nach der SubBytes-Schicht von AES, wenn die Zustandsmatrix

$$\begin{pmatrix} 09 & 09 & 09 & 09 \\ 09 & . & . & . \\ 09 & . & . & . \\ 09 & . & . & . \end{pmatrix}.$$

(alle Elemente sind gleich dem Byte 09 in Hex-Darstellung) in SubBytes eingegeben wird?

Ermitteln Sie dieses Ergebnis zunächst mathematisch (Invertieren in $GF(2^8)$ und danach affine Abbildung) und kontrollieren Sie danach mithilfe der Tabelle (Figure 7 in FIPS197). Hinweis: $(x^3 + 1)(x^6 + x^3 + x^2 + x + 1) = 1$ modulo $m(x) = x^8 + x^4 + x^3 + x + 1$.

20. **Shift-Rows-Transformation von AES:** Was ist das Ergebnis (der Zustand) nach der Shift-Rows-Schicht von AES, wenn die Zustandsmatrix

$$\begin{pmatrix} 2E & 63 & 63 & 63 \\ 2E & 63 & 63 & 63 \\ 2E & 63 & 63 & 63 \\ 2E & 63 & 63 & 63 \end{pmatrix}$$

ist?

Ermitteln Sie das Ergebnis mithilfe der AES Dokumentation FIPS197.

21. **MixColumns-Transformation von AES:** Was ist das Ergebnis (der Zustand) nach der MixColumns-Schicht von AES, wenn die Zustandsmatrix

$$\begin{pmatrix} 03 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \end{pmatrix}$$

in MixColumns eingegeben wird? Ermitteln Sie das Ergebnis mithilfe der AES Dokumentation FIPS197.

22. **MixColumns-Transformation von AES:** Was ist das Ergebnis (der Zustand) nach MixColumns, wenn die Zustandsmatrix

$$\begin{pmatrix} 00 & 00 & 00 & 00 \\ 0E & 0E & 0E & 0E \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \end{pmatrix}$$

in MixColumns eingegeben wird? Ermitteln Sie das Ergebnis mithilfe der AES Dokumentation FIPS197.

23. Was ist das Ergebnis (der Zustand) nach MixColumns, wenn die Zustandsmatrix

$$\begin{pmatrix} 00 & 00 & 01 & 00 \\ 00 & 00 & 00 & 00 \\ 03 & 01 & 00 & 01 \\ 01 & 00 & 00 & 01 \end{pmatrix}$$

in MixColumns eingegeben wird?

24. **Ergebnis einer Runde von AES**: Was ist das Ergebnis (der Zustand) nach der ersten Runde von AES, wenn der Eingang der ersten Runde (nach Verknüpfung mit k_0) die Zustandsmatrix

$$\begin{pmatrix} 09 & 52 & 52 & 52 \\ 09 & 52 & 52 & 52 \\ 09 & 52 & 52 & 52 \\ 09 & 52 & 52 & 52 \end{pmatrix}.$$

ist und der Rundenschlüssel k_1 aus lauter Einsen besteht? Führen Sie also hintereinander die 4 Schichten SubBytes, ShiftRows, MixColumns und AddRoundKey aus. Verwenden Sie dabei einfachheitshalber für SubBytes die Substitutionstabelle (S-Box in Figure 7, FIPS197).

25. **Betriebsmodi einer Blockchiffre**: Gegeben ist die einfach Blockchiffre E, die einen 4-Bit-Block $b_1b_2b_3b_4$ nach folgender Vorschrift verschlüsselt:

$$E(b_1b_2b_3b_4) = b_2b_3b_4b_1.$$

Weiters ist der Klartext 1011 0001 1011 gegeben, also drei Blöcke der Länge 4. Verschlüsseln Sie
a) im CFB-Mode b) OFB-Mode c) CTR-Mode und entschlüsseln Sie danach jeweils wieder. Verwenden Sie als Initialisierungsvektor jeweils $IV = 1100$ bzw. für CTR verwenden Sie $IV = 11$ als Initialisierungsvektor und hängen die Blöcke (counters) $00, 01, 10$ an.
Wie wirkt sich ein falsch übertragenes Bit im ersten Geheimtextblock bei der Entschlüsselung auf die Klartextblöcke aus?

26. **Geburtstagsparadoxon**:
a) Berechnen Sie: Wie groß ist die Wahrscheinlichkeit, dass unter n Studierenden (mindestens) zwei am selben Tag Geburtstag haben? (Nehmen Sie an, dass ein Jahr 365 Tage hat und dass jeder Tag als Geburtstag gleich wahrscheinlich ist.)
b) Berechnen Sie diese Wahrscheinlichkeit konkret für $n = 30$.

Lösungen zu den Aufwärmübungen

1. Ausgang $y = 000010 \ldots 0$ (alle 64 Bits sind 0 bis auf das Bit Nr. 5).

2. $E(r) = 000001010 \ldots 0$ (alle 48 Bits sind 0 bis auf die Bits Nummer 6 und 8.)

3. Das höchstwertige und das niedrigstwertige Bit bestimmen in binärer Form die Zeilennummer, also hier $(10)_2 = 2$. Die inneren vier Bits bestimmen die Spaltennummer, hier $1001 = 9$. Somit ist der Input 110010 auf den Tabelleneintrag in Zeile Nummer 2 und Spalte Nummer 9 der S_1-Tabelle (S_1-Box) abzubilden. In der S_1-Tabelle steht in Zeile Nummer 2 und Spalte Nummer 9 die Zahl 12. Diese ist, binär geschrieben, der gesuchte Ausgang.

$$S_1(110010) = 12 = 1100.$$

Aus Gründen der Übersichtlichkeit kann der Ausgang auch hexadezimal angegeben werden, also hier C.
Die S_1-Box befindet sich in der Feistel-Box (neben $S_2, \ldots, S_8$) als Teil der nichtlinearen Abbildung f. Sie wird in jeder Runde verwendet.

4. $S_1(000001)$ = Element aus Tabelle S_1 in Zeile Nummer 1 und Spalte Nummer 0: $S_1(000001) = 0$
   $S_2(010000)$ = Element aus Tabelle S_2 in Zeile Nummer 0 und Spalte Nummer 8: $S_2(010000) = 9$
   Somit erste 8 Bits des Ausgangs: 0000 1001

5. - Eingangspermutation IP: $y_1 = IP(x) = IP(0\ldots0) = 0\ldots0$ (Ergebnis lauter 0).
   - Feistel-Box: $F_{k_1}(y_1) =?$
     $y_1 = 0\ldots0$ wird in zwei Hälften geteilt:
     $l = 0\ldots0$, $r = 0\ldots0$ (linke Hälfte nur 0, rechte Hälfte nur 0)
     Die rechte Hälfte wird unter f abgebildet: $f(r, k_1) =?$
     Expansionspermutation: $E(r) = E(0\ldots0) = 0\ldots0$ (alle 48 Bit der Ergebnisses sind 0).
     X-or mit Teilschlüssel $k_1 = 0\ldots0$: ändert nichts
     S-Box: $S(0\ldots0) = 1110\,1111\,1010\,0111\,0010\,1100\,0100\,1101 = EFA72C4D$
     P-Box:

Input in P-Box	1110	1111	1010	0111	0010	1100	0100	1101
Out	1101	1000	1101	1000	1101	1011	1011	1100
Out (hex)	D	8	D	8	D	B	B	C

     also $f(r, k_1) = D8D8DBBC$.
     X-or mit linker Hälfte: ändert nichts, da linke Hälfte $l = 0\ldots0$.
     Somit $F_{k_1}(y_1) = D8D8DBBC00000000$ (hexadezimal dargestellt).
   - Vertauschung von Links und Rechts:
     $\Theta(D8D8DBBC00000000) = 00000000D8D8DBBC$

6. a) $03 \cdot 03 = (x+1)(x+1) = x^2 + 1 = 05$
   b) $04 + 02 = x^2 + x = 06$
   c) $03 \cdot 04 = (x+1)x^2 = x^3 + x^2 = 0C$
   d) $8C \cdot 02 = (x^7+x^3+x^2) \cdot x = x^8+x^4+x^3 = x+1 (\mathrm{mod}\,(x^8+x^4+x^3+x+1)) = 03$.

7. $$\begin{pmatrix} 2E & 63 & 63 & 63 \\ 2E & 63 & 63 & 63 \\ 2E & 63 & 63 & 63 \\ 2E & 63 & 63 & 63 \end{pmatrix}$$
   denn aus der Figure 7 in FIPS197 auf Seite 16 entnimmt man: $S(C3) = 2E$, $S(00) = 63$.

8. - a) Multiplikatives Inverses jedes Bytes: $(01)_{16} = (00000001)_2$ entspricht dem konstanten Polynom 1. Das multiplikative Inverse von 1 ist 1 selbst

(ist immer so;-). Damit erhalten wir zunächst

$$\begin{pmatrix} 00 & 00 & 00 & 01 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 \end{pmatrix}.$$

- Bytewise affine Transformation: für das Byte $(01)_{16} = (00000001)_2$ ergibt sich

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix},$$

also $(01111100)_2 = (7C)_{16}$. Für das Byte 00 erhalten wir $(01100011)_2 = (63)_{16}$. Der Zustand nach SubBytes ist damit

$$\begin{pmatrix} 63 & 63 & 63 & 7C \\ 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \end{pmatrix}.$$

b) Nach Figure 7 auf Seite 16 in FIPS197 ergibt sich $7C$.

9. Die 2., 3., und 4. Spalte bleiben unverändert; 1.Spalte:

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} 00 \\ 03 \\ 00 \\ 02 \end{pmatrix} = \begin{pmatrix} 03 \cdot 03 + 02 \cdot 01 \\ 03 \cdot 02 + 02 \cdot 01 \\ 03 \cdot 01 + 02 \cdot 03 \\ 03 \cdot 01 + 02 \cdot 02 \end{pmatrix} = \begin{pmatrix} 07 \\ 04 \\ 05 \\ 07 \end{pmatrix}.$$

Ergebnis von MixColumns ist daher

$$\begin{pmatrix} 07 & 00 & 00 & 00 \\ 04 & 00 & 00 & 00 \\ 05 & 00 & 00 & 00 \\ 07 & 00 & 00 & 00 \end{pmatrix}.$$

10.

$$\begin{pmatrix} A1 & 2F & 01 & 05 \\ 2F & 01 & 05 & A1 \\ 01 & 05 & A1 & 2F \\ 05 & A1 & 2F & 01 \end{pmatrix}$$

11. Wir verwenden die Notation aus Beispiel 4.16. Durch Einsetzten in unsere Beziehung
$$X_1^1 + X_2^1 + X_4^1 + Y_1^1 = 0$$
wird daraus
$$x_1 + x_4 + x_5 + x_7 + y_7 = k_1 + k_4 + k_7$$
Da die Gleichungen für X^j und Y^j immer richtig sind, ändern sich die Wahrscheinlichkeiten nicht und diese Beziehung ist ebenfalls in 81.25% aller Fälle richtig.

12. Bemerkung: In der Praxis ist $E = E_k$, es geht also noch ein geheimer Schlüssel ein! (z.B. E = AES)
Die Umkehrfunktion zu $E(b_1b_2b_3b_4) = b_2b_3b_4b_1$ (= das erste Bit wird an die letzte Stelle gesetzt) ist $E^{-1}(b_1b_2b_3b_4) = b_4b_1b_2b_3$ (= das letzte Bit wird an die erste Stelle gesetzt).
a) ECB: Verschlüsselt wird mit $y_j = E(x_j)$: $E(1011) = 0111$, $E(0001) = 0010$, $E(1011) = 0111$; entschlüsselt wird mit $x_j = E^{-1}(y_j)$: $E^{-1}(0111) = 1011$, $E^{-1}(0010) = 0001$, $E^{-1}(0111) = 1011$.
b) CBC: Verschlüsselt wird mit $y_j = E(x_j \oplus y_{j-1})$, wobei $y_0 = 1100$ gegeben ist: $y_1 = E(x_1 \oplus y_0) = E(1011 \oplus 1100) = E(0111) = 1110$, analog $y_2 = 1111$, $y_3 = 1000$. Entschlüsselt wird mit $x_j = E^{-1}(y_j) \oplus y_{j-1}$, z.B. Entschlüsselung von y_1: $E^{-1}(y_1) \oplus y_0 = E^{-1}(1110) \oplus 1100 = 0111 \oplus 1100 = 1011$, analog werden y_2 und y_3 entschlüsselt.

Lösungen zu ausgewählten Aufgaben

1. $0 \ldots 010 \ldots 0$ mit 1 an Stelle 36

2. $00001010 \ldots 0$ mit 1 an den Stellen 5 und 7

3. $40A72C4D$

4. $y = 10000000985099AC$

5. –

6. Der DES reduziert sich auf eine reine Permutation.

7. 2^{59} Bytes (ca. 576.5 Petabytes).

8. a) Beide Eigenschaften sind für allgemeines $\mathbf{a} = (a_1, a_2, a_3, a_4, a_5, a_6)$ und $\mathbf{b} = (b_1, b_2, b_3, b_4, b_5, b_6)$ aus $\mathbb{Z}_2^6$ zu beweisen.

b) $A = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$

9. $NOT(x_1) = 1 + x_1$ (affin), $OR(x_1, x_2) = x_1 + x_2 + x_1x_2$ (nichtlinear), $AND(x_1, x_2) = x_1x_2$ (nichtlinear), $XOR(x_1, x_2) = x_1 + x_2$ (linear).

10. Nein

11. $$\begin{pmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{pmatrix}$$

12.

0	y_1	y_2	$y_1 + y_2$
0	-0.25	0.25	0
x_1	0.25	-0.25	0.5
x_2	0.25	0.25	0
$x_1 + x_2$	0.25	0.25	0

Aus der letzten Spalte sehen wir $x_1 + y_1 + y_2 = 0$.

13. Nein

14. Nein. Es bietet keine Sicherheit gegen einen Klartextangriff. Auch zwei S-Boxen ändern daran nichts. Mit Key-Whitening ist das nicht mehr möglich.

15. Gegenbeispiel z.B. $S_1(000000) \neq 0000$.

16.

00000 00000	10000 10111	01100 01000	11100 11111
00011 00000	10011 10111	01111 01000	11111 11111

17. $x_4 + x_6 + x_7 + x_8 + y_8 = k_6 + k_8 + k_{10}$.

18. $\begin{pmatrix} 63 & 63 & 63 & 2E \\ 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \end{pmatrix}$.

19. $\begin{pmatrix} 01 & 01 & 01 & 01 \\ 01 & . & . & . \\ 01 & . & . & . \\ 01 & . & . & . \end{pmatrix}$.

20. –

21. $\begin{pmatrix} 06 & 00 & 00 & 00 \\ 03 & 00 & 00 & 00 \\ 03 & 00 & 00 & 00 \\ 05 & 00 & 00 & 00 \end{pmatrix}$

22. $\begin{pmatrix} 12 & 12 & 12 & 12 \\ 1C & 1C & 1C & 1C \\ 0E & 0E & 0E & 0E \\ 0E & 0E & 0E & 0E \end{pmatrix}$

23. $\begin{pmatrix} 02 & 03 & 02 & 00 \\ 04 & 02 & 01 & 02 \\ 05 & 01 & 01 & 01 \\ 01 & 01 & 03 & 03 \end{pmatrix}$

24. $\begin{pmatrix} FD & FE & FE & FC \\ FE & FE & FC & FD \\ FE & FC & FD & FE \\ FC & FD & FE & FE \end{pmatrix}$

25. a) $y_1, y_2, y_3 = 0010, 0101, 0001$; begrenzte Fehlerausbreitung
    b) $y_1, y_2, y_3 = 0010, 0010, 1101$; keine Fehlerausbreitung
    c) $y_1, y_2, y_3 = 0010, 1010, 0110$; keine Fehlerausbreitung

26. a) $P(n) = 1 - \frac{365 \cdot 364 \cdot \ldots (365-n+1)}{365^n}$ b) $P(30) = 70.6\%$

Literatur

[Bar20] E. Barker. „Recommendation for Key Management: Part 1 – General“. In: *NIST Special Publication* 800-57 Part 1, Revision 5 (2020). URL: https://doi.org/10.6028/NIST.SP.800-57pt1r5.

[Ber+11] G. Bertoni u. a. *Cryptographic sponge functions*. Techn. Ber. 2011. URL: https://keccak.team/files/CSF-0.1.pdf.

[BGW01] N. Borisov, I. Goldberg und D. Wagner. „Intercepting mobile communications: the insecurity of 802.11“. In: *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking* (2001), S. 180–189. URL: https://dl.acm.org/doi/10.1145/381677.381695 (besucht am 29. 03. 2019).

[BR19] E. Barker und A. Roginsky. „Transitioning the Use of Cryptographic Algorithms and Key Lengths“. In: *NIST Special Publication* 800-131A, Revision 2 (2019). URL: https://doi.org/10.6028/NIST.SP.800-131Ar2.

[BS91] E. Biham und A. Shamir. „Differential Cryptanalysis of DES-like Cryptosystems“. In: *Journal of Cryptology* 4 (1 1991), S. 3–72. URL: https://doi.org/10.1007/BF00630563.

[BS93] E. Biham und A. Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. New York: Springer, 1993. URL: http://www.cs.technion.ac.il/~biham/Reports/differential-cryptanalysis-of-the-data-encryption-standard-biham-shamir-authors-latex-version.pdf.

[BSI] Bundesamt für Sicherheit in der Informationstechnik (BSI). *Kryptographische Verfahren: Empfehlungen und Schlüssellängen*. Techn. Ber. 2023. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf (besucht am 09. 01. 2023).

[Cop94] D. Coppersmith. „The Data Encryption Standard (DES) and its strength against attacks“. In: *IBM J. RES. DEVELOP.* 38 (3 1994). URL: https://doi.org/10.1147/rd.383.0243 (besucht am 29. 03. 2019).

[DR01] J. Daemen und V. Rijmen. *The Design of Rijndael*. Berlin: Springer, 2001. URL: https://cs.ru.nl/~joan/papers/JDA_VRI_Rijndael_2002.pdf.

[Dwo07] M. Dworkin. „Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC“. In: *NIST Special Publication* 800-38D (2007). DOI: 10.6028/NIST.SP.800-38D. URL: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf.

[ENISA] *Cryptographic protocols and tools*. URL: https://www.enisa.europa.eu/topics/cryptography (besucht am 09. 03. 2023).

[FIPS180-4] NIST. „Secure Hash Standard (SHS)“. In: *Federal Information Processing Standards Publication (FIPS)* 180-4 (2015). URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf.

[FIPS197] NIST. „Advanced Encryption Standard (AES)“. In: *Federal Information Processing Standards Publication (FIPS)* 197 (2001). URL: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf.

[FIPS202] NIST. „SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions“. In: *Federal Information Processing Standards Publication (FIPS)* 202 (2015). URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf.

[FIPS46-3] NIST. „Data Encryption Standard (DES)“. In: *Federal Information Processing Standards Publication (FIPS)* 46-3 (1999). URL: https://csrc.nist.gov/csrc/media/publications/fips/46/3/archive/1999-10-25/documents/fips46-3.pdf (besucht am 19. 05. 2005).

[Hay02] H. M. Hays. „A tutorial on linear and differential cryptanalysis“. In: *Cryptologia* 26 (3 2002), S. 189–221. URL: https://doi.org/10.1080/0161-110291890885.

[Mat93] M. Matsui. „Linear Cryptanalysis Method for DES Cipher“. In: *Advances in Cryptology – EUROCRYPT '93* 765 (1993), S. 386–397. URL: https://doi.org/10.1007/3-540-48285-7_33.

[NIST01] NIST. „Recommendation for Block Cipher Modes of Operation“. In: *NIST Special Publication* 800-38A (2001). URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf.

[NIST12] NIST. „Recommendation for Applications Using Approved Hash Algorithms“. In: *NIST Special Publication* 800-107r1 (2012). URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-107r1.pdf.

[NIST17] NIST. *Block Cipher Techniques*. URL: https://csrc.nist.gov/projects/block-cipher-techniques (besucht am 09. 03. 2023).

[RFC1321] R. Rivest. „The MD5 Message-Digest Algorithm“. In: *Request for Comments* 1321 (1992). URL: https://www.rfc-editor.org/rfc/rfc1321 (besucht am 19. 02. 2018).

[RFC2104] H. Krawczyk, M. Bellare und R. Canetti. „HMAC: Keyed-Hashing for Message Authentication“. In: *Request for Comments* 2104 (1997). URL: https://www.rfc-editor.org/rfc/rfc2104 (besucht am 19. 02. 2018).

[RFC8018] K. M. Moriarty, B. Kaliski und A. Rusch. „PKCS #15: Password-Based Cryptography Specification“. In: *Request for Comments* 8018 (2017). URL: https://www.rfc-editor.org/rfc/rfc8018.

[Sch96] E. F. Schaefer. „A Simplified Data Encryption Standard Algorithm“. In: *Cryptologia* 20 (1 1996), S. 77–84. URL: https://doi.org/10.1080/0161-119691884799.

[Sha49] C. E. Shannon. „Communication Theory of Secrecy Systems“. In: *Bell System Technical Journal* 28.4 (Okt. 1949), S. 656–715. DOI: 10.1002/j.1538-7305.1949.tb00928.x.

[STE17] M. Stevens u. a. „The first collision for full SHA-1“. In: *IACR Cryptology ePrint Archive* 190 (2017). URL: https://shattered.io/static/shattered.pdf.

[TDEA17] E. Barker und N. Mouha. „Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher“. In: *NIST Special Publication* 800-67 Revision 2 (2017). URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-67r2.pdf.