Schlüsseln angewendet werden, der Aufwand liegt also in der Größenordnung von 2^{112}. Dreifacher DES oder, wie man meist sagt, Triple DES, gilt daher im Augenblick noch als sicher: Nachdem die DES-Cracker-Attacke publik geworden war, zog die amerikanische Regierung die Zulassung von DES für weniger geheime Nachrichten im Regierungsbereich zurück und verlangte stattdessen Triple DES, genauso wurden in Deutschland die Euroscheckkarten ersetzt durch neue Karten, die auf Triple DES anstelle des einfachen DES beruhen.

Soweit bekannt, bietet ein dreifacher DES mit drei verschiedenen Schlüsseln keine zusätzliche Sicherheit gegenüber einem mit nur zwei Schlüsseln, wobei die erste und die dritte Verschlüsselung denselben Schlüssel benutzen. Meist wird Triple DES in folgender Weise angewandt: Ein Block wird mit dem ersten Schlüssel *ver*schlüsselt, mit dem zweiten *ent*schlüsselt und dann noch einmal mit dem ersten *ver*schlüsselt.

§5: Operationsmodi

Bislang haben wir DES nur betrachtet für die Verschlüsselung eines einzelnen Blocks; tatsächlich besteht eine Nachricht aber meist aus einer ganzen Folge $x_1 x_2 \dots x_n$ von Blöcken. In diesem Paragraphen wollen wir uns überlegen, wie diese Nachricht am besten verschlüsselt wird.

Die Betrachtungen hier beziehen sich nicht speziell auf DES, sondern gelten genauso auch für jede andere Blockchiffre. Daher gehen wir hier aus von *irgendeiner* Blockchiffre

$$B\colon S \times X \to X; \quad (s,x) \mapsto B(s,x)\,,$$

die einem Block x in Abhängigkeit von einem Schlüssel s den Chiffretext $B(s,x)$ zuordnet.

a) Electronic Code Book (ECB)

Die naheliegendste Weise, eine Nachricht $x_1 x_2 \dots x_N$ zu verschlüsseln besteht darin, ihr die Nachricht $y_1 y_2 \dots y_N$ zuzuordnen mit $y_i = B(s,x_i)$, d.h jeder Block wird vor der Übertragung mit s verschlüsselt.

So einfach diese Methode auch ist, in der Praxis sollte man sie besser nicht anwenden. Ihre große Schwäche von ECB liegt in der Tatsache begründet, daß gleiche Klartextblöcke *immer* zu gleichen Chiffretextblöcken führen. Das bedeutet zwar nicht unbedingt, daß gleiche Klartextteile stets gleich verschlüsselt werden, denn wegen der Blockstruktur der Chiffre hängt der Chiffretext ja auch noch davon ab, wie weit der Textbeginn vom Blockanfang entfernt ist. Bei DES mit ASCII gibt es dafür aber nur acht Möglichkeiten, bei DES mit Unicode sogar nur vier, so daß bei längeren Texten in denen gewisse Namen und/oder Begriffe häufig vorkommen, durchaus die Gefahr einer größeren Anzahl identischer Chiffretextblöcke besteht.

Auch *magic bytes*, die bei vieler Dateiformaten als Dateianfang vorgeschrieben sind, führen stets zur selben Verschlüsselung; bei anderen Dateiformaten wie etwa ausführbaren Programmen oder gewissen Büroprogrammen gibt es innerhalb der Datei viele Blöcke von Nullen, *usw.*, so daß jemand, der alle Nachrichten eines Absenders abhört die Empfänger leicht in Klassen einteilen kann, die (ungefähr) dieselbe Information erhalten. Dadurch kennt er zwar noch nicht den Inhalt der Nachrichten, kann aber vielleicht doch sehr nützliche Informationen gewinnen.

Manchmal kann ein Gegner auch einfach dadurch Schaden anrichten, daß er unbemerkt die Reihenfolge von Nachrichtenblöcken vertauscht oder aber einen Nachrichtenblock *mehrfach* übermittelt. Er könnte auch eine neue Nachricht generieren, die aus Teilen bereits übermittelter Nachrichten zusammengesetzt ist; falls er die Struktur der Nachrichten auf Grund gemeinsamer Blöcke erkennt, hat er sogar eine gute Chance, daß die entstehende Nachricht sinnvoll ist. Bei Nachrichten mit festem Format, wie sie beispielsweise im elektronischen Zahlungsverkehr unter Banken üblich sind, hätte er eventuell sogar die Möglichkeit, zwei von ihm selbst initiierte Transaktionen zu identifizieren und zu seinem Vorteil zu manipulieren. Aber auch das bloße Einschleusen einer nicht als falsch zu erkennenden Nachricht etwa zu Sabotagezwecken kann bereits genügend Schaden anrichten.

Natürlich hat ein Angreifer bei einer guten Blockchiffre auch im ECB-Modus keine Chance, die Nachricht zu dechiffrieren oder gar den

Schlüssel herauszufinden, aber wie wir gesehen haben, kann er sich bei gewissen Typen von Nachrichten doch einiges an Information verschaffen.

Man kann in der Kryptographie üblicherweise nicht davon ausgehen, daß ein Anwender über die Stärken und Schwächen des verwendeten Kryptosystems Bescheid weiß: Er verläßt sich darauf, daß das gekaufte oder von einem Experten eingerichtete System seine Geheimnisse zuverlässig schützt, egal worum es sich handelt. Daher sollte man den ECB-Modus im Normalfall nicht benutzen.

b) Cipher Block Chaining (CBC)

Hier wird die Nachricht $x_1 x_2 \dots x_N$ übermittelt als $y_1 y_2 \dots y_N$ mit

$$y_i = B(s, x_i \oplus y_{i-1}) .$$

Da es für $i = 1$ noch keinen Chiffretextblock y_{i-1} gibt, muß dieser Anfangsblock explizit festgelegt werden.

Unabhängig von der Wahl des Anfangsblock hängt bei CBC jeder übertragene Block y_i auch noch vom Vorgänger y_{i-1} ab; es ist daher nicht möglich, eine Nachricht durch Auslassen von Blöcken oder durch Zusammensetzen zweier existierender Nachrichten zu manipulieren ohne daß Blöcke verfälscht und damit unentschlüsselbar werden – was den Empfänger (hoffentlich) zum Nachfragen veranlaßt.

Ein weiterer Vorteil besteht darin, daß jeder übertragene Block y_i von *jedem* der Blöcke $x_1, x_2, \dots, x_i$ abhängt; insbesondere hängt als der letzte Block y_N von jedem einzelnen Klartextblock ab. Falls also die übermittelte Nachricht noch elektronisch unterschrieben werden soll, reicht es, den Block y_N zu unterschreiben.

(Mit elektronischen Unterschriften werden wir uns im Zusammenhang mit der asymmetrischen Kryptographie beschäftigen. Wir werden dann auch Verfahren kennenlernen, wie man auch bei anderen Übertragungsmodi oder gar der Übertragung von Klartext einen Block finden kann, der von der gesamten Nachricht abhängt. Einen solchen Block bezeichnet man als *message authentication code*, kurz MAC. Allgemein berechnet man ihn durch sogenannte sichere Hash-Verfahren; eine gute

Blockchiffre im CBC-Modus liefert ein, wenn auch vergleichsweise aufwendiges, solches Verfahren.)

Die Abhängigkeit eines jeden Chiffreblocks von allen vorausgehenden Klartextblöcken hat nicht nur Vorteile: Sie führt auch dazu, daß Übertragungsfehler nicht nur einen Block betreffen. Tatsächlich führen sie aber bei CBC nur zur falschen Entschlüsselung zweier Blöcke: Die Entschlüsselungsfunktion ist bei CBC offenbar

$$x_i = B^{-1}(s, y_i) \oplus y_{i-1} ,$$

bereits $x_{i+2} = B^{-1}(s, y_{i+2}) \oplus y_{i+1}$ ist also von einen falsch übermittelten Block y_i nicht mehr betroffen.

Ein großes Problem beim ECB-Modus war, daß gleiche Nachrichten und auch gleiche Blöcke gleich übermittelt werden. Beim CBC-Modus ist dieses Problem zumindest insofern abgemildert, als gleiche Block durch das XOR mit dem vorangegangenen Chiffreblock verschieden chiffriert werden. Falls man allerdings den Anfangsblock y_0 konstant wählt – aus Sicht des Anwenders sicherlich die einfachste Lösung – werden identische Nachrichten weiterhin identisch chiffriert.

Die Sicherheit wird also auf jeden Fall erhöht, wenn für jede Übertragung ein neuer Anfangsblock benutzt wird. Dieser könnte beispielsweise ein Zufallsblock sein, der – damit ihn auch der Empfänger kennt – entweder unverschlüsselt oder ECB-verschlüsselt als erstes übertragen wird. Damit wird die zu übermittelnde Nachricht um einen Block verlängert, was im allgemeinen kein großes Problem ist – außer vielleicht in dem Fall, daß man sehr viele sehr kurze Nachrichten über eine teure oder stark kapazitätsbeschränkte Leitung übertragen muß.

Ein zufälliger Anfangsblock hilft noch nicht gegen das Problem, daß ein Angreifer einfach eine aufgefangene Nachricht ein zweites Mal in die Leitung einspielt. Da so etwas beispielsweise bei elektronischen Finanztransfers unbedingt erkannt werden muß, enthalten entsprechende Nachrichten selbstverständlich eine eindeutige Buchungsnummer.

Auch in anderen Systemen ist es oft üblich, daß jede Nachricht ihre eindeutige Kennzeichnung hat, und das legt es nahe, zumindest in solchen Systemen entweder direkt diese Nachrichtennummer oder aber

eine daraus abgeleitete Zahl (eine sogenannte *Nonce;* die Bezeichnung ist eine Kontraktion von *N*umber used *once*) zu verwenden. Da auch Nachrichtennummern Informationen enthalten, sollte diese Nummer zur Sicherheit mit der Blockchiffre verschlüsselt werden.

Ganz perfekt ist die Chiffre auch so noch nicht: Angenommen, der Chiffretextblock y_i ist gleich dem Block y_j. Dann können wir wie folgt argumentieren:

$$y_i = B(s, x_i \oplus y_{i-1}) \wedge y_i = B(s, x_j \oplus y_{j-1}) \Longrightarrow x_i \oplus y_{i-1} = x_j \oplus y_{j-1} \Longrightarrow x_i \oplus x_j = y_i \oplus y_j \ .$$

Somit läßt sich die Differenz $x_i \oplus x_j$ aus der Differenz der vorangegangenen Chiffretextblöcke $y_{i-1} \oplus y_{j-1}$ berechnen. Falls die Nachrichtenquelle eine ähnlich hohe Redundanz hat wie die deutsche Sprache, sollte diese Information ausreichen, um die beiden Blöcke (bis auf Reihenfolge) zu rekonstruieren.

Dies ist sicherlich ein Schwachpunkt, den man in der besten aller Welten gerne vermeiden würde; andererseits ist die Wahrscheinlichkeit, daß zwei gleiche Chiffretextblöcke auftreten, nicht sonderlich groß: Wenn wir davon ausgehen, daß sich Chiffretext im CBC-Modus wie eine Zufallsfolge verhält (was wahrscheinlich etwas zu optimistisch ist), liegt sie im Falle der Blocklänge n etwa bei $2^{-n/2}$. Bei einer 64-Bit-Blockchiffre wie DES heißt das, daß wir etwa 2^{32} = 4294967296 oder rund 4,3 Milliarden Blöcke brauchen, bevor wir mit einer Wahrscheinlichkeit von mindestens 50% zwei gleiche Chiffretextblöcke finden. Bei einer Blockchiffre mit 128 Bit (was heute eigentlich Mindeststandard sein sollte, kommt man sogar auf $2^{64} \approx 1{,}8 \cdot 10^{19}$ oder rund 18 Trilliarden Blöcke. (Für Einzelheiten sei auf das Kapitel über sichere Hashverfahren verwiesen, wo wir das sogenannte *Geburtstagsparadoxon* genauer betrachten werden.)

Da wir in Wirklichkeit natürlich keine Zufallsfolge haben, dürften die tatsächlichen Wahrscheinlichkeiten wohl etwas größer sein, aber bei normalen Textdateien sollten sie weiterhin praktisch vernachlässigbar sein, und bei wirklich großen Dateien wie etwa Videofilmen sollte die Kenntnis einiger weniger einzelner Blöcke für einen Angreifer wohl nutzlos sein. Was bleibt, ist das Restrisiko, daß beispielsweise genau der eine Block, in dem ein besonders streng geheimzuhaltender Name

oder Begriff steht zufälligerweise trotzdem genauso verschlüsselt wird wie ein anderer Block und damit einem ohne große Erfolgsaussichten auf genau dieses Restrisiko hoffenden Gegner bekannt wird – dies gehört zum unvermeidbaren Risiko eines jeden nicht absolut sicheren Kryptosystems.

Als Randbemerkung sollte erwähnt werden, daß obige Rechnung natürlich auch zeigt, daß verschiedene Klartextblöcke zu verschiedenen Chiffretextblöcken führen, falls die Chiffretextblöcke in den Vorgängerpositionen verschieden sind. Dies mag zwar auf den ersten Blick als nicht sehr informativ erscheinen, aber die Enigma wurde im zweiten Weltkrieg geknackt eben wegen der Beobachtung, daß sie nie einen Buchstaben durch sich selbst verschlüsselt. Bei einer Blockchiffre von 64 oder 128 Bit kann man mit so einer Information zwar sehr viel weniger anfangen, aber es handelt sich doch Information für den Gegner, von der wir nicht sicher sein können, was er damit anfangen kann.

c) Cipher Feedback (CFB)

Die nun folgenden Modi sind nützlich, wenn Daten in Echtzeit übertragen werden sollen, die kürzer sind als die Blocklänge; hier verwenden wir die Blockchiffre, um einen Schlüsselstrom zu erzeugen, der nach Art des *one time pad* verwendet wird. Der große Unterschied ist natürlich, daß die Entropie dieses Schlüsselstroms nur die des Schlüssels und des (ähnlich zu CBC verwendeten) Anfangsblocks ist: Unser guter alter Feind, der BAYESsche Gegner, hätte also keinerlei Schwierigkeiten, die Chiffre zu entschlüsseln. Unsere Hoffnung und der publik gewordene Teil der Erfahrung im Umgang mit Blockchiffren wie DES und AES beruht darauf, daß der Schlüsselstrom zu komplex ist für einen realen Gegner.

Bei CFB gehen wir davon aus, daß die Daten nicht als Blöcke anfallen, sondern in eventuell kleineren Einheiten zu k Bit. Typisch für Anwendungen ist der Wert k = 8, d.h. wir verschlüsseln einen Strom von Bytes, aber selbst der Fall k = 1, bei dem einzelne Bits verschlüsselt werden, kommt gelegentlich vor. Falls k kleiner ist als die Blocklänge n des Codes, ist dieser Modus also um den Faktor n/k langsamer als die bislang betrachteten Modi.

Auch hier gehen wir aus von einem Anfangsblock; er ist allerdings durch k fast vollständig festgelegt: In einem Register$R$, dessen Länge gleich der Blocklänge des verwendeten Codes ist, stehen rechts k Bit, zum Beispiel lauter Einsen, die restlichen Bits des Registers werden auf Null gesetzt. Sodann werden die *ersten* k Bit von $B(s, R)$ zu den ersten k Bit der Nachricht addiert und dies wird übertragen. Man beachte, daß das Verschlüsselungsergebnis nur für die Übertragung benutzt wird; der Inhalt des Registers behält seinen Wert.

In jedem der folgenden Schritt wird der Inhalt des Registers um k Bit (nichtzyklisch) nach links verschoben, und die k zuletzt übertragenen Bits werden am rechten Ende eingesetzt. Sodann werden die ersten k Bit des mit dem neuen Registerinhalt berechneten Blocks $B(s, R)$ zum nächsten Nachrichtenblock addiert und übertragen, *usw.*

Sofern die ersten k Bit, die ins Register geschrieben werden, konstant sind, werden gleiche Texte stets gleich verschlüsselt, und – was schlimmer ist – zum ersten Block der Nachricht wird stets derselbe Schlüssel addiert, so daß die statistischen Angriffe aus dem ersten Kapitel anwendbar sind. Falls k gleich der Blocklänge ist, könnte ein damit erfolgreicher Angreifer sogar den Wert $B(s, R)$ für den Anfangszustand des Registers rekonstruieren und, zumindest im Falle DES mit Hilfe von DES-Cracker oder einem ähnlichen Werkzeug den Schlüssel s ermitteln. Hier liefert also die Wahl eines deutlich unterhalb der Blocklänge liegenden Werts von k einen zusätzlichen Sicherheitsfaktor. Bei einer guten und zeitgemäßen Blockchiffre ist es natürlich unmöglich, aus einem Paar von Klar- und Chiffretextblöcken den Schlüssel zu rekonstruieren – es sei denn, man verfügt über die Rechenkraft des BAYESschen Gegners.

Auch in der praktischen Anwendung gibt es ein Problem, denn wie bei CBC hängt wieder jedes übertragene Wort aus k Bit von allen Vorgängern ab. Aus Sicht des Empfängers allerdings hängt der Inhalt des Registers nur ab von den letzten r empfangenen Chiffretextblöcken, wobei r die kleinste ganze Zahl ist mit $rk \geq n$, denn nach r Übertragungen fällt jeder Chiffreblock wegen der zyklischen Verschiebung aus dem Register heraus. Ein Übertragungsfehler beeinflußt hier also insgesamt $r + 1$ Nachrichtenblöcke.

d) Output feedback (OFB)

Typische Anwendungen von Stromchiffren sind Satellitenübertragungen. Hier sind Bitfehler auf Grund atmosphärischer Störungen relativ häufig; obwohl sie natürlich durch fehlerkorrigierende Codes so weit wie möglich kompensiert werden, muß man doch immer wieder mit auch längerfristigen erhöhten Fehlerraten rechnen. Dabei ist die Eigenschaft des CFB-Modus, jeden Fehler gleich auf $r + 1$ Blöcke durchschlagen zu lassen, höchst unwillkommen.

Ein für solche Anwendungen nützlicher Modus ist *output feedback* (OFB). Auch dieser Modus erzeugt einen Schlüsselstrom mit Hilfe eines Registers R, allerdings hängt dessen Inhalt weder vom Klartext noch vom Chiffretext ab. Da der Schlüsselstrom zum Nachrichtenstrom addiert wird, betrifft daher ein Bitfehler bei der Übertragung hier nur ein einziges Bit.

Das Register R wird zu Beginn auf einen Anfangswert gesetzt. Im Gegensatz zu CFB wird das Register selbst in jedem Schritt verschlüsselt, sein Inhalt also durch $B(s, R)$ ersetzt. Danach werden die ersten k Bit zum Nachrichtenblock addiert, und vor dem nächsten Schritt wird das Register *zyklisch* um k Positionen nach links verschoben.

Bei dieser Vorgehensweise *muß* man natürlich für jede Übertragung einen neuen Anfangsblock und/oder Schlüssel wählen, denn ansonsten wird mehrfach derselbe Schlüsselstrom verwendet, ein Gegner kann also schon mit einer relativ kleinen Anzahl von Chiffretexten durch Häufigkeitsanalysen Informationen über den Klartext bekommen, die bis zur völligen Entschlüsselung führen können. Da die Blockchiffre hier nur zur Erzeugung eines Schlüsselstroms verwendet wird, ist die zu betrachtende Einheit aus Sicht des Kryptanalytikers kein Block, sondern die kleinste Dateneinheit der Nachricht, typischerweise also ein Byte, so daß die Verfahren aus dem ersten Kapitel problemlos angewandt werden können. Außerdem muß darauf geachtet werden, daß der Schlüsselstrom natürlich periodisch ist. Die Periode ist zwar, abgesehen von einigen wenigen sogenannten *schwachen* Schlüsseln der Blockchiffre, sehr groß, aber je nach zu übertragendem Datenvolumen kann es trotzdem Probleme geben.

e) Counter mode (CTR)

Dieser Modus wird im Standard für DES nicht erwähnt, wurde aber 2001 vom NIST (dem *National Institute of Standards* der Vereinigten Staaten) als eine Methode zur Anwendung von Blockchiffren standardisiert.

Ausgangspunkt ist eine *Nonce,* d.h. eine Zahl a, die für genau eine Nachricht und danach nie wieder während der Gültigkeitsdauer des Schlüssels verwendet wird. Sie wird beispielsweise aus der Nummer oder dem Übertragungsdatum der Nachricht nach einem vorher definierten Verfahren erzeugt.

An diese Zahl wird wie bei OFB ein von der Nachricht unabhängiger Schlüsselstrom erzeugt, hier nach der Vorschrift

$$s_i = B(s, a||i)\,,$$

wobei $a||i$ für eine Vorschrift steht, wie die Blocknummer i hinter die Zahl a geschrieben wird. Konkret geht es also darum, daß a eine gewisse maximale Bitlänge hat, und die restlichen Bits werden für i reserviert.

Wie bei OFB *muß* auch hier natürlich sichergestellt sein, daß keine zwei Blöcke mit demselben s_i verschlüsselt werden, d.h. die Anzahl möglicher Werte für i muß größer sein als die maximale Länge einer zu übertragenen Nachricht. Bei 64 Bit-Chiffren kann dies den Wertebereich von i deutlich einschränken; bei einer Blocklänge von 128 Bit sollte es jedoch mit realistischen Nachrichten keine Probleme geben.

Die Verschlüsselung geschieht auch hier wie beim *one time pad,* d.h.

$$y_i = x_i \oplus s_i = x_i \oplus B(s, a||i)\,.$$

Auch hier muß wieder unbedingt sichergestellt werden, daß derselbe Schlüsselstrom nur einmal benutzt wird, d.h. die Zahl a darf auf keinen Fall mehrfach benutzt werden, da sonst die Attacken aus dem ersten Kapitel greifen würden.

Sofern dies wirklich sichergestellt ist, dürfte CTR wohl der sicherste unter den hier diskutierten Modi sein.

§6: Literatur

Da DES rund 25 Jahre lang *das* Standardverfahren für ernsthafte symmetrische Verschlüsselung im zivilen Bereich war, ist er natürlich in praktisch jedem Lehrbuch der Kryptologie ausführlich beschrieben, z.B. in

JAN C.A. VAN DER LUBBE: *Basic Methods of cryptography,* Cambridge University Press, 1998

oder

JOHANNES BUCHMANN: *Einführung in die Kryptographie,* Springer, ^{3}2003

Eine ausführliche Diskussion der Operationsmodi findet man unter anderem in

A.J. MENEZES, P.C. VAN OORSCHOT, S.A. VANSTONE: *Handbook of applied cryptography,* CRC Press 1997

sowie in

NIELS FERGUSON, BRUCE SCHNEIER: *Practical Cryptography,* Wiley, 2003

Speziellere Fragen sind außer in den bereits im Text zitierten Arbeiten und Büchern auch in fast jeder Konferenz über Kryptologie behandelt; insbesondere gilt dies für Tagungen wie *Crypto, Eurocrypt* und *Asiacrypt,* deren Proceeding jeweils in den Springer Lecture Notes in Computer Science erscheinen.

Kapitel 4
Der Advanced Encryption Standard Rijndael

§ 1: Geschichte und Auswahlkriterien

DES wurde in Zusammenarbeit mit der National Security Agency der Vereinigten Staaten von IBM entwickelt und dann als amerikanischer Standard verkündet. Diese Vorgehensweise weckte von Anfang an den Verdacht, daß möglicherweise eine „Falltür“ eingebaut sei, insbesondere da zumindest ursprünglich nicht alle Design-Kriterien publiziert wurden.

Nachdem dreißig Jahre intensiver Kryptanalyse keine Falltür gefunden haben, müßte diese – so vorhanden – zumindest sehr gut versteckt sein; aber egal ob mit oder ohne Falltür: Wie wir im letzten Kapitel gesehen haben, erfüllt DES mit nur 56 Bit Schlüssellänge nicht mehr die heutigen Sicherheitsanforderungen. Ursprünglich war er ohnehin nur für eine Laufzeit von zehn Jahren vorgesehen und wurde nur immer wieder verlängert, weil die meisten Anwender von Kryptographie äußerst träge sind und sich nicht um Fortschritte der Kryptanalyse kümmern.

Obwohl es die Internationale Standardisierungsorganisation ISO ablehnt, ein Kryptoverfahren zu standardisieren (Ein Grund dafür ist die dann befürchtete Bündelung krimineller Energie auf dieses Verfahren), hat das das amerikanische Handelsministerium die Suche nach einem Nachfolgealgorithmus für DES am 2. Januar 1997 international ausgeschrieben; der vorläufige Name des Algorithmus war AES *(Advanced Encryption Standard)*. Federführend für die Auswahl war das *National Institute of Standards and Technology* (NIST) in Gaithersburg, Maryland.

Dieses ging von Anfang an davon aus, daß der zu wählende Algorithmus *stärker* sein müsse als Triple DES; er sollte zwanzig bis dreißig Jahre lang anwendbar sein und dementsprechende Sicherheit bieten. Nach einer internationalen Konferenz über die Auswahlkriterien am 15. April 1997 veröffentlichte es am 12. September 1997 die endgültige Ausschreibung.

Minimalanforderung an die einzureichenden Algorithmen waren danach, daß es sich um symmetrische Blockchiffren handeln muß, die mindestens eine Blocklänge von 128 Bit bei Schlüssellängen von 128 Bit, 192 Bit und 256 Bit vorsieht.

Als Kriterien für die Wahl zwischen den einzelnen Algorithmen wurden die folgenden Aspekte genannt:

1. **Sicherheit:** Wie sicher ist der Algorithmus im Vergleich zu den anderen Kandidaten? Inwieweit ist seine Ausgabe ununterscheidbar von der einer Zufallspermutation? Wie gut ist die mathematische Basis für die Sicherheit des Algorithmus begründet? (Im Gegensatz zu DES sollten dieses Mal alle Kriterien publiziert werden.)
2. **Kosten:** Welche Lizensgebühren werden fällig? Wie effizient geht der Algorithmus mit Rechenzeit und Speicherplatz um?
3. **Flexibilität:** Ein Algorithmus, der auf einer Vielzahl von Plattformen implementierbar ist (PCs, 8-Bit-Prozessoren, ATM-Netze, HDTV, . . .) ist vorzuziehen, genauso einer, der auch als Stromchiffre, kryptographisch sichere Hash-Funktion oder ähnliches verwendet werden kann.
4. **Software:** Der Algorithmus *muß* auch softwaremäßig effizient implementierbar sein.
5. **Einfachheit:** Der Aufbau des Algorithmus soll möglichst einfach sein.

Nicht nur die Art der Suche unterschied sich also beträchtlich von der DES-Entwicklung hinter verschlossenen Türen, auch die Auswahlkriterien hatten sich stark geändert: DES war noch in erster Linie für Hardware entworfen worden; manche Aspekte von DES wie etwa die für die kryptographische Sicherheit völlig irrelevante Anfangspermutation hatten wohl keinen anderen Sinn als die Verlangsamung von Software-

Implementierungen.

Inzwischen hat freilich die Erfahrung mit der Kryptographie in der Geschäftswelt gezeigt, daß man dort den Kostenaufwand für Spezialhardware scheut und sich lieber mit den dubiosen Verfahren begnügt, die in der ohnehin vorhandenen Office-Software eingebaut ist (Preis für das Knacken durch spezialisierte Unternehmen je nach Programm zwischen 40 $ und 250 $), wohingegen sich die Hacker innerhalb und außerhalb der Geheimdienste durch einen hohen Aufwand nur wenig beeindrucken lassen.

Die Ausschreibung führte bis zum Abgabeschluß am 15. Juni 1998 zu fünfzehn Vorschlägen aus aller Welt. Diese wurden auf einer ersten AES-Konferenz vom 20.–22. August 1998 in Ventura, Kalifornien vorgestellt und der Fachwelt zur Analyse und Kommentierung empfohlen. Tatsächlich wurden vierzehn der fünfzehn Algorithmen schon vor der Konferenz veröffentlicht; lediglich der Algorithmus *Magenta* der Deutschen Telekom wurde erst am 20. August auf der Konferenz selbst bekanntgegeben. Er war auch der einzige Algorithmus, der bereits während der Konferenz geknackt wurde in einer auf den 20. August 1998 datierten Arbeit von E. BIHAM, A. BIRYUKOV, N. FERGUSON, L. KNUDSEN, B. SCHNEIER und A. SHAMIR, die noch auf der Konferenz verteilt wurde.

Die zweite AES-Konferenz am 22. und 23. April 1999 in Rom führte zu einer ersten Diskussion der Ergebnisse und Empfehlungen, welche der fünfzehn Algorithmen weiter betrachtet werden sollten. Die endgültige Entscheidung des NIST wurde am 9. August 1999 bekanntgegeben:

Bei fünf der eingereichten Algorithmen hatte sich herausgestellt, daß sie entweder mit einem Aufwand zu knacken sind, der deutlich unter der Durchsuchung des gesamten Schlüsselraums liegt (darunter natürlich auch Magenta) oder aber, daß es zu viele „schwache“ Schlüssel gibt. Da diese fünf Algorithmen auch mit die langsamsten Kandidaten waren, sprach nichts dafür, sie noch weiter zu betrachten.

Fünf weitere Kandidaten zeigten ebenfalls Schwächen, die zwar für sich allein betrachtet nicht so schwerwiegend waren, daß man die Verfahren

eliminieren müßte; da diese fünf Kandidaten andererseits nirgends entscheidende Vorteile boten, wurden auch sie eliminiert, so daß noch fünf Finalisten übrigblieben: Drei aus USA (MARS von IBM, RC6 von RSA und Twofish von Counterpane), Serpent von drei Kryptologen aus England, Israel und Norwegen, und Rijndael von zwei Kryptologen, die von der Katholischen Universität Leuven in Belgien kommen.

Bei der dritten AES-Konferenz am 13. und 14. April 2000 in New York bekam Rijndael 86 Stimmen, Serpent 59, Twofish 31, RC6 23 und MARS 13, so daß es nicht weiter verwunderte, daß das NIST am 2. Oktober 2000 Rijndael für den vorgeschlagenen Standard nominierte. Als offizieller Federal Information Processing Standard FIPS-197 verkündete ihn das amerikanische Handelsministerium am 6. Dezember 2001.

Rijndael hat seinen Namen von seinen beiden Autoren JOAN DAEMEN und VINCENT RIJMEN, die 1995 beziehungsweise 1997 an der Elektrotechnischen Fakultät der Katholischen Universität Leuven über Themen aus der Kryptographie promoviert hatten; RIJMEN arbeitet immer noch dort als Postdoc. Aufgrund der Wahl von Rijndael zum AES wurden sie als flämische Persönlichkeiten des Jahres 2000 gewählt.

Als Aussprachehilfe für Personen, die kein Niederländisch, Flämisch, Surinamer oder Afrikaans sprechen, geben die Autoren folgende englische Approximationen des Wortes „Rijndael“: „Reign Dahl“, „Rain Doll“ und „Rhine Dahl“.

§ 2: Algebraische Vorbereitungen

Alle Operationen von Rijndael sind algebraisch definiert. Es gibt einmal Operationen auf Byte-Ebene, die durch die Grundrechenarten im Körper mit 256 Elementen realisiert sind; dazu kommen Operationen auf 4-Byte-Wörtern, die im Polynomring über diesem Körper definiert sind. Wir wollen uns langsam an die Definition dieses Körpers und dieses Ringes herantasten; im Hinblick auf Algorithmen der *public key*-Kryptographie, die uns später in der Vorlesung interessieren werden, ist die Präsentation geringfügig allgemeiner, als es für Rijndael notwendig wäre.

a) Der Euklidische Algorithmus für natürliche Zahlen

Wir werden den EUKLIDischen Algorithmus im folgenden hauptsächlich für Polynome brauchen; zum besseren Verständnis der Idee (und für spätere Anwendungen in der *public key*-Kryptographie) betrachten wir ihn aber zunächst im wohlbekannten Fall der natürlichen Zahlen.

Ziel ist die Berechnung des größten gemeinsamen Teilers (ggT) zweier natürlicher Zahlen a und b, also der größten natürlichen Zahl d, die sowohl a als auch b teilt; wir schreiben kurz

$$d = \mathrm{ggT}(a, b)\,.$$

Grundidee des EUKLIDischen Algorithmus ist die Anwendung der Division mit Rest: Für je zwei natürliche Zahlen x und y gibt es nichtnegative ganze Zahlen q und r, so daß

$$x = qy + r \quad \text{und} \quad 0 \le r < y$$

ist. Alsdann ist

$$\mathrm{ggT}(x, y) = \mathrm{ggT}(y, r)\,,$$

denn wegen der beiden Gleichungen

$$x = qy + r \quad \text{und} \quad r = x - qy$$

teilt jeder gemeinsame Teiler von x und y auch r, und jeder gemeinsame Teiler von y und q teilt x.

Der EUKLIDische Algorithmus nutzt dies aus, um die Zahlen, deren ggT bestimmt werden muß, sukzessive zu verkleinern, bis der ggT zweier Zahlen berechnet werden muß, von denen die eine Teiler der anderen ist; in diesem Fall ist natürlich die kleinere der beiden Zahlen gleich dem ggT.

Formal sieht der EUKLIDische Algorithmus zur Berechnung des ggT zweier natürlicher Zahlen a und b folgendermaßen aus:

Schritt 0: Setze $r_0 = a$ und $r_1 = b$

Schritt i, $i \ge 1$**:** Falls $r_i = 0$ ist, endet der Algorithmus mit $\mathrm{ggT}(a, b) = r_{i-1}$; andernfalls dividiere man r_{i-1} mit Rest durch r_i mit dem Ergebnis $r_{i-1} = q_i r_i + r_{i+1}$.

(Bei einer tatsächlichen Implementierung bieten sich hier natürlich einige offensichtliche Optimierungen an.)

Der Algorithmus muß nach endlich vielen Schritten enden, denn bei der Division mit Rest ist stets $0 \le r_{i+1} < r_i$, so daß r_i mit jedem Schritt kleiner wird, was bei natürlichen Zahlen nicht unbegrenzt möglich ist. Da außerdem in jedem Schritt

$$\mathrm{ggT}(r_{i-1}, r_i) = \mathrm{ggT}(r_i, r_{i+1})$$

ist und im letzten Schritt, wenn r_{i-1} den vorigen Wert r_{i-2} teilt,

$$\mathrm{ggT}(r_{i-1}, r_{i-2}) = r_{i-1}$$

ist, folgt induktiv

$$\mathrm{ggT}(a, b) = r_{i-1}\,,$$

so daß der Algorithmus das richtige Ergebnis liefert.

Wir brauchen für das folgende vor allem eine Erweiterung des Grundalgorithmus, den sogenannten erweiterten EUKLIDischen Algorithmus: Die Gleichung

$$r_{i-1} = q_i r_i + r_{i+1}$$

läßt sich umschreiben als

$$r_{i+1} = r_{i-1} - q_i r_i\,,$$

so daß r_{i+1} eine ganzzahlige Linearkombination von r_i und r_{i-1} ist. Da entsprechend auch r_i Linearkombination von r_{i-1} und r_{i-2} ist, folgt induktiv, daß der ggT von a und b als ganzzahlige Linearkombination von a und b dargestellt werden kann.

Algorithmisch sieht diese Erweiterung folgendermaßen aus:

Schritt 0: Setze $r_0 = a$, $r_1 = b$, $\alpha_0 = \beta_1 = 1$ und $\alpha_1 = \beta_0 = 0$. Mit $i = 1$ ist dann

$$r_{i-1} = \alpha_{i-1} a + \beta_{i-1} b \quad \text{und} \quad r_i = \alpha_i a + \beta_i b\,.$$

Diese Relationen werden in jedem der folgenden Schritte erhalten:

Schritt i, $i \ge 1$**:** Falls $r_i = 0$ ist, endet der Algorithmus mit

$$\mathrm{ggT}(a, b) = r_{i-1} = \alpha_{i-1} a + \beta_{i-1} b\,.$$

Andernfalls dividiere man r_{i-1} mit Rest durch r_i mit dem Ergebnis

$$r_{i-1} = q_i r_i + r_{i+1} .$$

Dann ist

$$\begin{aligned} r_{i+1} = q_i r_i - r_{i-1} &= q_i(\alpha_i a + \beta_i b) - (\alpha_{i-1} a + \beta_{i-1} b) \\ &= (q_i \alpha_i - \alpha_{i-1}) a + (q_i \beta_i - \beta_{i-1}) b ; \end{aligned}$$

man setze also

$$\alpha_{i+1} = q_i \alpha_i - \alpha_{i-1} \quad \text{und} \quad \beta_{i+1} = q_i \beta_i - \beta_{i-1} .$$

Genau wie oben folgt, daß der Algorithmus für alle natürlichen Zahlen a und b endet und daß am Ende der richtige ggT berechnet wird; außerdem sind die α_i und β_i so definiert, daß in jedem Schritt $r_i = \alpha_i a + \beta_i b$ ist, insbesondere ist also im letzten Schritt der ggT als Linearkombination der Ausgangszahlen dargestellt.

Als Beispiel wollen wir den ggT von 200 und 148 als Linearkombination darstellen. Im nullten Schritt haben wir 200 und 148 als die trivialen Linearkombinationen

$$200 = 1 \cdot 200 + 0 \cdot 148 \quad \text{und} \quad 148 = 0 \cdot 200 + 1 \cdot 148 .$$

Im ersten Schritt dividieren wir, da 148 nicht verschwindet, 200 mit Rest durch 148:

$$200 = 1 \cdot 148 + 52 \quad \text{und} \quad 52 = 1 \cdot 200 - 1 \cdot 148 .$$

Da auch $52 \neq 0$, dividieren wir im zweiten Schritt 148 durch 52:

$148 = 2 \cdot 52 + 44$ und $44 = 148 - 2 \cdot (1 \cdot 200 - 1 \cdot 148) = 3 \cdot 148 - 2 \cdot 200$.

Auch $44 \neq 0$; wir machen also weiter: $52 = 1 \cdot 44 + 8$ und

$8 = 52 - 44 = (1 \cdot 200 - 1 \cdot 148) - (3 \cdot 148 - 2 \cdot 200) = 3 \cdot 200 - 4 \cdot 148$.

Im nächsten Schritt erhalten wir $44 = 5 \cdot 8 + 4$ und

$4 = 44 - 5 \cdot 8 = (3 \cdot 148 - 2 \cdot 200) - 5 \cdot (3 \cdot 200 - 4 \cdot 148) = 23 \cdot 148 - 17 \cdot 200$.

Bei der Division von acht durch vier schließlich ist der Divisionsrest null; damit ist vier der ggT von 148 und 200 und kann in der angegebenen Weise linear kombiniert werden.

Auch wenn wir es im Augenblick nicht ganz so allgemein brauchen, sei zumindest kurz erläutert, wie der erweiterte EUKLIDische Algorithmus zum Lösen diophantischer Gleichungen verwendet werden kann, von Gleichungen also, bei denen nur ganzzahlige Lösungen interessieren. Wir betrachten nur die einfache lineare Gleichung

$$ax + by = c \quad \text{mit} \quad a, b, c \in \mathbb{Z}$$

für zwei Unbekannte $x, y \in \mathbb{Z}$.

Der größte gemeinsame Teiler $d = \text{ggT}(a, b)$ von a und b teilt offensichtlich jeden Ausdruck der Form $ax + by$ mit $x, y \in \mathbb{Z}$; falls d kein Teiler von c ist, kann es also keine ganzzahlige Lösung geben.

Ist aber $c = rd$ ein Vielfaches von d und ist

$$d = \alpha a + \beta b$$

die lineare Darstellung des ggT nach dem erweiterten EUKLIDischen Algorithmus, so ist offensichtlich

$$x = r\alpha \quad \text{und} \quad y = r\beta$$

eine Lösung.

Ist (x', y') eine weitere Lösung, so ist

$$a(x - x') + b(y - y') = c - c = 0 \qquad \text{oder} \qquad a(x - x') = b(y' - y) .$$

$v = a(x - x') = b(y' - y)$ ist also ein gemeinsames Vielfaches von a und b und damit auch ein Vielfaches des kleinsten gemeinsamen Vielfachen von a und b. Dieses kleinste gemeinsame Vielfache ist ab/d, es muß also eine ganze Zahl m geben mit

$$x - x' = m \cdot \frac{b}{d} \quad \text{und} \quad y' - y = m \cdot \frac{a}{d} .$$

Die allgemeine Lösung der obigen Gleichung ist somit

$$x = r\alpha - m \cdot \frac{b}{d} \ \text{ und } \ y = r\beta + m \cdot \frac{a}{d} \quad \text{mit} \quad m \in \mathbb{Z} .$$

b) Körper von Primzahlordnung

Die Addition und Multiplikation im Ring $\mathbb{Z}$ der ganzen Zahlen induziert für jede natürliche Zahl n eine Addition und Multiplikation in

$$\mathbb{Z}/n \underset{\text{def}}{=} \{0, 1, \ldots, n-1\}$$

durch

$$a \oplus b \underset{\text{def}}{=} (a + b) \bmod n \quad \text{und} \quad a \otimes b = (a \cdot b) \bmod n \,.$$

Im allgemeinen macht dies jedoch $\mathbb{Z}/n$ nicht zum Körper, denn ist $n = p \cdot q$ Produkt zweier Faktoren, die beide kleiner als n und damit größer als eins sind, so ist $p \odot q = 0$, obwohl weder p noch q gleich null sind. Solche sogenannte *Nullteiler* können in einem Körper nicht vorkommen, denn ist in einem Körper $x \cdot y = 0$ und $x \neq 0$, so folgt

$$y = x^{-1} \cdot x \cdot y = y^{-1} \cdot 0 = 0 \,.$$

Für zusammengesetzte Zahlen n ist $\mathbb{Z}/n$ daher kein Körper.

Ist allerdings $n = p$ eine Primzahl, so läßt sich jedes von null verschiedene Element aus $\mathbb{Z}/p$ invertieren: Für jede natürliche Zahl a mit $0 < a < p$ ist der ggT von a und p gleich eins, so daß es nach dem erweiterten EUKLIDischen Algorithmus ganze Zahlen α und β gibt mit

$$\alpha a + \beta p = 1 \qquad \text{oder} \qquad \alpha a - 1 = \beta p \,.$$

Damit ist $\alpha a - 1 = \beta p$ durch p teilbar oder, anders ausgedrückt

$$\alpha a \equiv 1 \bmod p \,.$$

Somit ist $\alpha \bmod p$ ein multiplikatives Inverse zu a.

Wenn wir auf der Menge $\mathbb{F}_p \underset{\text{def}}{=} \{0, 1, \ldots, p-1\}$ eine Addition und Multiplikation einführen durch die Vorschriften

$$a \oplus b \underset{\text{def}}{=} (a + b) \bmod p \quad \text{und} \quad a \odot b \underset{\text{def}}{=} ab \bmod p$$

definieren, ist klar, daß – genau wie in den ganzen Zahlen – das Kommutativ- und das Assoziativgesetz sowohl für die $\oplus$ als auch für $\odot$ gilt, und auch das Distributivgesetz folgt sofort aus dem für $\mathbb{Z}$. Bezüglich $\oplus$ ist die Null neutrales Element und $p - a$ invers zu a; bezüglich $\odot$ ist

die Eins neutrales Element, und wie wir gerade gesehen haben, gibt es zu jedem $a \neq 0$ ein Inverses. Somit ist $\mathbb{F}_p$ ein Körper.

Das Rechnen in diesem Körper ist größtenteils einfach: Die Addition kann auf die gewöhnliche Addition in $\mathbb{Z}$ zurückgeführt werden; da $a + b$ für $a, b \in \mathbb{F}_p$ zwischen Null und $2p - 2$ liegt, ist

$$a \oplus b = \begin{cases} a + b & \text{falls } a + b < p \\ a + b - p & \text{sonst} \end{cases} \,.$$

Bei der Multiplikation ist die Situation nicht ganz so einfach; hier braucht man eine Division mit Rest, um $a \oplus b$ zu berechnen.

Das additive Inverse ist, wie bereits erwähnt, einfach $p - a$; die Berechnung des multiplikativen Inversen erfordert einen erweiterten EUKLIDischen Algorithmus und ist damit die rechenaufwendigste Operation in $\mathbb{F}_p$.

Bei der Kryptographie mit öffentlichen Schlüssel werden uns Körper $\mathbb{F}_p$ zu großen Primzahlen p interessieren; hier, für AES, brauchen wir nur den Fall $p = 2$, in dem natürlich alles trivial ist:

$\oplus$	0	1
0	0	1
1	1	0

$\odot$	0	1
0	0	0
1	0	1

c) Euklidische Ringe

Für den EUKLIDischen Algorithmus im ersten Abschnitt waren im wesentlichen zwei Dinge wesentlich: Ersten mußten wir wissen, wann eine Zahl Teiler einer anderen ist, und zweitens brauchten wir eine Divison mit Rest, für die der Rest in irgendeiner Weise kleiner als der Divisor ist, wobei jede Folge immer kleiner werdender Reste nach endlich vielen Schritten abbrechen muß. Diese beiden Eigenschaften werden im Begriff des EUKLIDischen Rings formalisiert:

Definition: *a)* Ein Ring ist eine Menge R zusammen mit zwei Verknüpfungen

$$+, \cdot : R \times R \to R \,,$$

für die gilt:

1.) $(R,+)$ ist eine abelsche Gruppe
2.) $a(bc) = (ab)c$ für alle $A, b, c \in R$
3.) $a(b+c) = ab + ac$ für alle $a, b, c \in R$.

b) R heißt *kommutativer Ring mit Eins,* falls zusätzlich gilt

4.) Es gibt ein Element $1 \in R$, so daß $1 \cdot a = a \cdot 1 = a$ für alle $a \in R$
5.) $a \cdot b = b \cdot a$ für alle $a, b \in R$.

c) Ein kommutativer Ring mit Eins heißt *Integritätsbereich,* wenn gilt:

6.) Für $a, b \in R \setminus \{0\}$ ist auch $ab \neq 0$.

d) Ein EUKLIDischer Ring ist ein Integritätsbereich R zusammen mit einer Abbildung $\nu\colon R \setminus \{0\} \to \mathbb{N}_0$, für die gilt:

7.) Zu je zwei Elementen $a, b \in R \setminus \{0\}$ gibt es $q, r \in R$ mit

$$a = bq + r \quad \text{und} \quad \nu(r) < \nu(b) \ \text{ falls } r \neq 0\,.$$

Offensichtlich ist der Ring $\mathbb{Z}$ der ganzen Zahlen ein EUKLIDischer Ring; hier können wir einfach $\nu(a) = |a|$ setzen. Als Übung kann man auch leicht zeigen, daß der Ring

$$\Gamma = \mathbb{Z} \oplus \mathbb{Z}i = \left\{a + bi \in \mathbb{C} \;\middle|\; a, b \in \mathbb{Z}\right\}$$

der GAUSSschen Zahlen EUKLIDisch ist mit

$$\nu(a + ib) = a^2 + b^2\,.$$

Interessanter für uns ist, daß für jeden Körper k der Polynomring

$$k[X] = \Big\{\sum_{\ell=0}^{n} a_\ell X^\ell \;\Big|\; n \in \mathbb{N}_0 \ \text{ und } \ a_i \in k\Big\}$$

ein EUKLIDischer Ring ist, wobei die Funktion ν hier einfach jedem Polynom seinen Grad zuordnet, also den Exponenten der höchsten vorkommenden X-Potenz. Hier zeigt die übliche Polynomdivison mit Rest, daß es in der Tat zu je zwei Polynomen f, g mit Koeffizienten aus k Polynome q, r gibt, so daß

$$f = gq + r \quad \text{und} \quad r = 0 \ \text{ oder } \ \deg r < \deg g\,.$$

Da EUKLIDische Ringe insbesondere Integritätsbereiche sind, können wir auch dort größte gemeinsame Teiler definieren:

Definition: *a)* Ein Element t eines Integritätsbereichs R heißt Teiler von $r \in R$, in Zeichen $t|r$, wenn es ein $q \in R$ gibt mit $r = qt$.
b) $d \in R$ heißt größter gemeinsamer Teiler von $r, s \in R$, wenn $d|r, s$ und wenn für jeden weiteren gemeinsamen Teiler t von r und s gilt: $t|d$.

Eine besondere Rolle spielen die Teiler der Eins: Für einen solchen Teiler t gibt es ein Element q mit $qt = 1$, d.h. t hat ein Inverses bezüglich der Multiplikation. Solche Elemente bezeichnen wir als *Einheiten;* offensichtlich bilden die Einheiten aus R eine multiplikative Gruppe, die sogenannte *Einheitengruppe* $R^\times$.

Man beachte, daß die so definierten größten gemeinsamen Teiler nicht eindeutig bestimmt sein müssen: In $\mathbb{Z}$ beispielsweise ist sowohl drei als auch minus drei ein größter gemeinsamer Teiler von sechs und minus neun. Allgemein gilt:

Lemma: In einem EUKLIDischen Ring R gibt es zu je zwei Elementen r, s, die nicht beide gleich null sind, einen größten gemeinsamen Teiler d. Dieser kann nach dem EUKLIDischen Algorithmus berechnet werden und läßt sich als Linearkombination

$$d = \alpha r + \beta s \quad \text{mit} \quad \alpha, \beta \in R$$

darstellen. Sind d und d' zwei größte gemeinsame Teiler von r und s, so gibt es eine Einheit e mit $d' = ed$.

Beweis: Die Existenz eines größten gemeinsamen Teilers folgt genau wie im Fall der natürlichen Zahlen: Ist $a = bq + r$, so ist ein Element $t \in R$ genau dann gemeinsamer Teiler von a und b, wenn es gemeinsamer Teiler von b und r ist. Daher gibt es genau dann einen größten gemeinsamen Teiler von a und b, wenn es einen größten gemeinsamen Teiler von b und r gibt, und dieser ist dann gleichzeitig größter gemeinsamer Teiler von a und b.

Der EUKLIDische Algorithmus kann genauso definiert werden wir im letzten Abschnitt, und genau wie dort muß er auch enden mit einem Divisionsrest null, denn die Folge der Zahlen $\nu(r_i) \in \mathbb{N}_0$ ist strikt fallend. Im Falle $r_n = 0$ ist ggT$(r_{n-2}, r_{n-1}) = r_{n-1}$, da r_{n-1} dann r_{n-2} ohne Rest teilt; induktiv folgt, daß das auch ein ggT von a und b ist.

Sind d und d' zwei größte gemeinsame Teiler von a und b, so sind beide insbesondere gemeinsame Teiler von a und b; nach Definition eines größten gemeinsamen Teilers muß daher d Teiler von d' sein und umgekehrt. Es gibt somit ein $e \in R$ mit $d' = ed$ und ein $e' \in R$ mit $d = e'd'$. Also ist

$$d = e'd' = e'ed \Longrightarrow (1 - e'e)d = 0 \Longrightarrow 1 - e'e = 0 \Longrightarrow e'e = 1\,,$$

da R nach Voraussetzung ein Integritätsbereich ist. Die letzte Gleichung rechts zeigt, daß e eine Einheit ist.

Schließlich müssen wir noch zeigen, daß sich *jeder* größte gemeinsame Teiler d von a und b als Linearkombination von a und b schreiben läßt. Der erweiterte EUKLIDische Algorithmus liefert eine solche Darstellung für *einen* größten gemeinsamen Teiler

$$d' = \alpha a + \beta b\,;$$

sind e, e' die oben betrachtete Einheit zu d und d', so folgt

$$d = e'd' = (\alpha e')a + (\beta e')b\,,$$

wie behauptet. ∎

Als Beispiel wollen wir für $k = \mathbb{Q}$ den größten gemeinsamen Teiler der beiden Polynome

$$P = X^8 + X^6 - 3X^4 - 3X^3 + 8X^2 + 2X - 5$$

und

$$Q = 3X^6 + 5X^4 - 4X^2 - 9X + 21$$

berechnen: Division von P durch Q führt auf den Quotienten $X^2/3 - 2/9$ und Divisionsrest

$$R_2 = -\frac{5}{9}X^4 + \frac{1}{9}X^2 - \frac{1}{3}\,.$$

Division von Q durch R_2 ergibt

$$R_3 = -\frac{117}{25}X^2 - 9X + \frac{441}{25}\,,$$

bei der Division von R_2 durch R_3 bleibt Rest

$$R_4 = \frac{233150}{6591}X - \frac{102500}{2197}\,,$$

und bei der letzten Divison verbleibt als Rest der ggT

$$R_5 = \frac{1288744821}{543589225}\,.$$

Da beide Ausgangspolynome ganzzahlige Koeffizienten haben, erscheint ein ggT mit einem so großen Nenner seltsam. Wir wissen aber, daß „der“ größte gemeinsame Teiler nur bis auf Einheiten bestimmt ist, und im Polynomring über einem Körper sind alle von null verschiedenen Konstanten Einheiten. Der größte gemeinsame Teiler ist daher nur eindeutig bis auf Multiplikation mit einer nichtverschwindenden Konstanten; diese Konstante kann nach Belieben gewählt werden und wird meist so gewählt, daß das Ergebnis in irgendeinem Sinne einfach wird.

Auf das obige Beispiel angewendet heißt das, daß mit

$$R_5 = \frac{1288744821}{543589225}$$

auch eins ein ggT von A und B ist und man daher im allgemeinen sagen würde, „der“ ggT von A und B sei eins. Es ist ein wohlbekanntes (und umgehbares) Problem der Computeralgebra, daß der EUKLIDische Algorithmus diese einfache Lösung in einer so komplizierten Form liefert; da wir aber vor allem Polynome über endlichen Körpern benötigen, braucht uns das nicht weiter zu kümmern.

d) Endliche Körper von Primzahlpotenzordnung

In Abschnitt *b)* hatten wir die ganzen Zahlen modulo einer Primzahl p zu einem Körper gemacht; der einzige nichttriviale Schritt dabei war die Existenz des multiplikativen Inversen, die wir aus der linearen Kombinierbarkeit des ggT folgerten und daraus, daß der ggT einer Zahl mit einer Primzahl gleich eins ist, falls die Zahl kein Vielfaches der Primzahl ist.

Genauso wollen wir jetzt Körper definieren, indem wir Polynome über einem festen Körper k modulo einem vorgegebenen Polynom P betrachten: Für ein beliebiges Polynom A über k ist A mod P gleich dem Rest bei der Division von A durch P.

Falls A kleineren Grad als P hat, ist natürlich einfach A mod $P = A$; zum konkreten Rechnen können wir daher ausgehen vom Vektorraum V

aller Polynome vom Grad höchstens d, wobei $d+1$ der Grad von P ist. Die Addition ist die gewöhnliche Addition von Polynomen, das Nullpolynom ist Neutralelement, und $-A$ ist invers zu A.

Das Produkt AB zweier Polynome $A, B \in V$ kann größeren Grad als d haben; wir setzen daher

$$A \odot B = AB \bmod P\,;$$

dies ist ein Polynom vom Grad höchstens d, und es ist klar, daß die so definierte Multiplikation kommutativ und assoziativ ist und das Distributivgesetz erfüllt. Das konstante Polynom 1 ist Neutralelement auch bezüglich dieser Multiplikation. Algebraisch gesehen identifizieren wir V also mit dem Faktorring $k[X]/(P)$.

Ein inverses Polynom zu A ist ein Polynom B, für das $A \odot B = 1$ ist, d.h.

$$AB = 1 + CP \qquad \text{oder} \qquad AB + CP = 1$$

für ein geeignetes Polynom C. Zu vorgegebenen Polynomen A und P gibt es solche Polynome B und C genau dann, wenn der ggT von A und P gleich eins ist; alsdann lassen sich B und C nach dem erweiterten EUKLIDischen Algorithmus berechnen.

Wenn wir möchten, daß jedes Polynom A, dessen Grad kleiner als deg P ist, ein Inverses hat, müssen wir sicherstellen, daß A und P immer teilerfremd sind; dies ist offensichtlich genau dann der Fall, wenn P keinen nichttrivialen Teiler hat, keinen Teiler also, dessen Grad größer als null und kleiner als deg P ist. Ein solches Polynom heißt *irreduzibel*.

Falls es ein irreduzibles Polynom P vom Grad n mit Koeffizienten aus k gibt, läßt sich der Vektorraum k^n also zu einem Körper machen, indem wir ein n-tupel $(a_0, \ldots, a_{n-1})$ mit dem Polynom

$$a_{n-1}X^{n-1} + a_{n-2}X^{n-2} + \cdots + a_1X + a_0$$

identifizieren und die Multiplikation als Multiplikation von Polynomen modulo P erklären.

Bekanntestes Beispiel ist $k = \mathbb{R}$: Für $n = 2$ gibt es irreduzible Polynome vom Grad n, beispielsweise das Polynom $P = X^2 + 1$. Da

$$\begin{aligned}(a_1X + a_0)(b_1X + b_0) &= a_1b_1X^2 + (a_0b_1 + a_1b_0)X + a_0b_0 \\ &\equiv (a_0b_1 + a_1b_0)X + (a_0b_0 - a_1b_1) \bmod X^2 + 1\end{aligned}$$

ist, folgt

$$(a_0, a_1) \odot (b_0, b_1) = (a_0b_0 - a_1b_1, a_0b_1 + a_1b_0)\,,$$

wir erhalten also den Körper der komplexen Zahlen. Weitere Beispiele über $\mathbb{R}$ gibt es nicht, denn ein irreduzibles reelles Polynom muß entweder Grad eins oder Grad zwei haben, und da jedes irreduzible quadratische Polynom zwei konjugiert komplexe Nullstellen hat, entstehen dabei immer die komplexen Zahlen – lediglich die Basis über $\mathbb{R}$ ändert sich.

Über endlichen Körpern ist die Situation etwas komplizierter: Hier gibt es für *jedes* n mindestens ein irreduzibles Polynom vom Grad n, allerdings gilt auch hier, daß zwei irreduzible Polynome desselben Grads auf den gleichen Körper führen. Eine kurze Beweisskizze ist unten angedeutet; einen vollständigen Beweis findet man in jedem Lehrbuch der Algebra.

Es gibt keinen einfachen Ausdruck für ein irreduzibles Polynom vom Grad n über einem vorgegebenen endlichen Körper k; in der Computeralgebra behilft man sich meist damit, daß man so lange zufällige Polynome vom Grad n erzeugt, bis man ein irreduzibles gefunden hat – ein Algorithmus, der sich in der Praxis als deutlich effizienter erweist als manch ein deterministischer Algorithmus zur Lösung von Standardproblemen.

Es gibt allerdings auch eine Alternative, die zumindest für kleine Körper durchaus anwendbar ist: Ist $k = \mathbb{F}_q$ ein endlicher Körper mit q Elementen, so ist $k \setminus \{0\}$ eine multiplikative Gruppe der Ordnung $q-1$. Da die Ordnung eines jeden Elements einer Gruppe Teiler der Gruppenordnung ist, folgt

$$x^{q-1} = 1 \quad \text{für alle } x \in \mathbb{F}_q \setminus \{0\}\,.$$

Dies gilt insbesondere für das Element X mod P, das $\mathbb{F}_q$ über $\mathbb{F}_p$ erzeugt, also ist P ein Teiler von $X^{q-1} - 1$. Die möglichen Polynome P

zur Erzeugung von $\mathbb{F}_{p^n}$ über $\mathbb{F}_p$ sind also genau die irreduziblen Teiler vom Grad n des Polynoms $X^{p^n} - 1$. Die Computeralgebra stellt gerade für Polynome über endlichen Körpern effiziente Faktorisierungsalgorithmen zur Verfügung, so daß man diese Teiler noch für recht große Werte von p^n relativ schnell berechnen kann.

Als weitere Konsequenz aus obiger Formel kommt man auch zu einer neuen Interpretation des Körpers mit $q = p^n$ Elementen: Da alle q-1 Elemente von $\mathbb{F}_q \setminus \{0\}$ Nullstellen des Polynoms $x^{q-1}-1$ sind und dieses Polynom den Grad $q-1$ hat, handelt es sich hier um *alle* Nullstellen von $x^{q-1}-1$; in der Sprache der Algebra ist $\mathbb{F}_q$ daher der Zerfällungskörper des Polynoms $x^{q-1}-1$ über $\mathbb{F}_p$. Daraus folgt wegen der Eindeutigkeit des Zerfällungskörpers, daß alle Körper mit q Elementen isomorph sind.

e) Der Körper mit 256 Elementen

Für AES ist der Körper $\mathbb{F}_{256}$ von zentraler Bedeutung; da $256 = 2^8$ ist, handelt es sich hier um den Vektorraum $\mathbb{F}_2^8$. Die Addition ist sehr einfach: Da die Addition in $\mathbb{F}_2$ mit dem exklusiven Oder übereinstimmt, ist die Addition in $\mathbb{F}_{256}$ das bitweise exklusive Oder, eine Operation, die nicht nur in den gängigen CPUs, sondern auch in vielen Prozessoren für spezielle Anwendungen in der Signalverarbeitung als Grundoperation implementiert und somit sehr schnell ist.

Um eine Multiplikation zu definieren, brauchen wir ein irreduzibles Polynom vom Grad acht über $\mathbb{F}_2$. Da $\mathbb{F}_2$ ein sehr kleiner Körper und acht eine ziemlich kleine Zahl ist, hat zumindest ein Computer keinerlei Schwierigkeiten, alle diese Polynome zu bestimmen: Wie wir im vorigen Abschnitt gesehen haben, sind das genau die irreduziblen Faktoren vom Grad acht in der Faktorisierung des Polynoms $X^{255} - 1$ über $\mathbb{F}_2$. Faktorisierung von Polynomen über Körpern von Primzahlordnung ist einer der Grundalgorithmen der Computeralgebra und geht auch noch bei sehr viel komplizierteren Polynomen sehr schnell; hier zeigt das Ergebnis, daß es dreißig Faktoren vom Grad acht gibt. Diese führen zwar alle auf denselben Körper, aber das praktische Rechnen in diesem Körper hängt natürlich stark von der Wahl des Polynoms ab. Insbesondere wird die Geschwindigkeit umso höher, je weniger Terme das Polynom hat.

Dreizehn der dreißig Polynome bestehen aus sieben nichtverschwindenden Termen, die restlichen siebzehn aus fünf; Wir wählen natürlich eines der letzteren. Alle diese Polynome haben, wie jedes Polynom vom Grad acht über $\mathbb{F}_2$, den führenden Term X^8; danach folgen vier weitere Terme. Bei der Reduktion modulo einem solchen Polynom $P = X^8 + \mathit{Rest}$ benutzt man, daß dann

$$X^8 \equiv \mathit{Rest}, \qquad X^9 \equiv X \cdot \mathit{Rest}, \quad \ldots$$

ist; dies wird umso häufiger mehrfach angewandt werden müssen, je höheren Grad die Terme in *Rest* haben. Am effizientesten kann man also rechnen, wenn das Polynom *Rest* den kleinstmöglichen Grad hat, und wenn zudem auch noch die hinteren Terme von *Rest* möglichst kleinen Grad haben. Inspektion der siebzehn Polynome mit fünf Termen zeigt, daß das Polynom

$$m(X) = X^8 + X^4 + X^3 + X + 1$$

in dieser Hinsicht optimal ist.

Die genaue Festlegung für das Rechnen in $\mathbb{F}_{256} = \mathbb{F}_2^8$ für die Zwecke von AES ist folgende: Wir schreiben ein Byte als $(a_7, a_6, \ldots, a_0)$ und identifizieren es mit dem Polynom

$$a_7X^7 + a_6X^6 + \ldots + a_1X + a_0\,;$$

das Byte 0000 0010 entspricht also X.

Der Einfachheit halber schreiben wir Bytes meist als zweiziffrige Hexadezimalzahlen: Im betrachteten Beispiel wäre das 02_{hex}, und das Byte $\text{A5}_{\text{hex}} = 1010\,0101$ entspricht dem Polynom $X^7 + X^5 + X^2 + 1$.

Man beachte, daß trotz dieser Schreibweise die Addition und Multiplikation in $\mathbb{F}_{256}$ natürlich nichts mit der Addition und Multiplikation von Hexadezimalzahlen zu tun haben. Zwar ist $01_{\text{hex}} + 02_{\text{hex}} = 03_{\text{hex}}$, aber $01_{\text{hex}} + 01_{\text{hex}} = 00_{\text{hex}}$ und $05_{\text{hex}} + 04_{\text{hex}} = 01_{\text{hex}}$.

Zur Berechnung von $\text{A5}_{\text{hex}} \odot 01_{\text{hex}}$ müssen wir das Polynom

$$(X^7 + X^5 + X^2 + 1) \cdot X = X^8 + X^6 + X^3 + X$$

berechnen und modulo $m(X)$ reduzieren. Da

$$X^8 \bmod m(X) = X^4 + X^3 + X^2 + 1$$