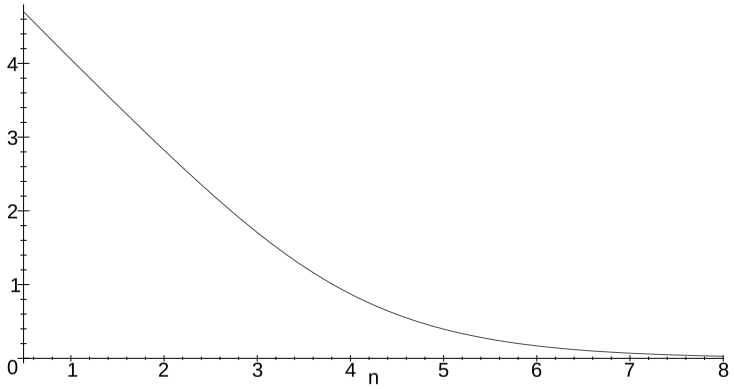


Abb. 2: Mehrdeutigkeit des Schlüssels via Kontakthäufigkeiten

## g) Folgen für Sicherheitsanforderungen

Unter dem Gesichtspunkt der Sicherheit von Kryptosystemen sind die Ergebnisse dieses Kapitels ziemlich deprimierend: Perfekte Sicherheit gibt es höchstens dann, wenn die Anzahl der möglichen Schlüssel mindestens genauso groß ist wie die der möglichen Klartexte und wenn die Schlüssel zudem auch noch perfekt zufällig gewählt werden. Dann allerdings gibt es, wie wir schon im ersten Kapitel gesehen haben, in der Tat ein einfaches und perfekt sicheres Verfahren: den *one time pad*.

Bei kürzeren Schlüsseln verliert man nicht nur die perfekte Sicherheit, sondern sehr schnell sogar jegliche Spur von Sicherheit gegen einen Gegner mit hinreichend großer Rechenkraft.

Als nächsten Schritt weg von idealen, aber schwer realisierbaren Sicherheitsstandards, könnten wir versuchen, von Gegnern mit beschränkten Möglichkeiten auszugehen. Das ist insofern unbedenklich, als daß die Möglichkeiten eines jeden realen Gegners in der Tat beschränkt sind – das Problem ist nur, daß uns unsere Gegner nichts über ihre Möglichkeiten (oder auch nur ihre Existenz) verraten, so daß wir ihre Grenzen nicht kennen. Wir kennen jedoch zumindest einigermaßen den Stand der auf dem offenen Markt erhältlichen Technik, und wenn auch einige unserer Gegner definitiv mit Mitteln arbeiten, die *nicht* auf dem offenen Markt erhältlich sind, können wir dies durch einen zusätzlichen Sicherheitsfaktor zu berücksichtigen versuchen. Gegen einen sensationellen Durchbruch in einem geheimen Labor sind wir dadurch zwar nicht gefeit, aber zumindest gegen einen „normal“ verlaufenden technischen Fortschritt der nächsten Jahre können wir uns schützen.



Unser Problem beschränkt sich dann also darauf, uns gegen einen Gegner zu schützen, dessen Rechenmöglichkeiten wir kennen. Wir brauchen somit nur noch ein Verfahren, von dem wir sicher sind, daß er es mit all seiner Hardware nicht in weniger als – je nach unseren Sicherheitsanforderungen – einem Jahr oder vielleicht auch hundert Jahren brechen kann.

Wenn wir wüßten, welche Verfahren der Gegner einsetzt, wäre das relativ einfach; leider wird es uns das aber nicht verraten. Wir können nicht ausschließen, daß selbst ein Gegner mit wenig Hardware und wenig Finanzmitteln plötzlich eine zündende Idee hat, eine uns bislang noch unbekannte Schwäche unseres Verfahrens auszunutzen. Wirklich sicher sind wir daher auch vor ihm nur, wenn wir den Rechenaufwand *aller* Algorithmen, die unser Kryptosystem für einen speziellen Schlüssel brechen, nach unten abschätzen können.

Dies ist in der Tat einfach möglich, aber leider völlig nutzlos: Angenommen, wir haben eine Schlüsselmenge $S$ und wählen daraus zufällig den Schlüssel $s_0 \in S$. Eine mögliche Strategie eines Gegners könnte darin bestehen, einfach einen (oder mehrere) feste Schlüssel aus $S$ durchzuprobieren. Wie der Erfolg sogenannter *dictionary attacks* zeigt, ist dies oft sogar eine recht intelligente Strategie. Falls sein Algorithmus eine Schlüsselmenge vorsieht, die $s_0$ enthält, hat er gewonnen.

Da wir im Gegensatz zu all den inkompetenten Anwendern realer Kryptosysteme unsere Schlüssel selbstverständlich rein zufällig bestimmen (zumindest, sobald wir wissen, was das bedeutet und wie was geht), hat er gegen uns damit natürlich keine große Chance – aber doch zumindest eine kleine.

Nehmen wir an, unsere Schlüsselmenge hat $2^{128}$ Elemente – das ist der Bereich, in dem seriöse Alltagskryptographie heute anfangen sollte, auch wenn die Sicherheiten unsere ec-Karten derzeit nur darauf beruht, daß niemand eine gewise 112-Bit-Zahl kennt. Dann liegt

die Wahrscheinlichkeit, daß ein Gegner ausgerechnet unser $s_0$ wählt, bei $2^{-128} \approx 0{,}29 \cdot 10^{-38}$. Zum Vergleich: Die Wahrscheinlichkeit für einen Sechser im Lotto liegt bei $\binom{49}{6}^{-1} \approx 0{,}72 \cdot 10^{-7}$; die Wahrscheinlichkeit für fünf aufeinanderfolgende Sechser im Lotto ist also immer noch etwa 636 mal höher als die einer korrekten Schlüsselwahl. Wenn wir uns den Aufwand für perfekte Sicherheit nicht leisten wollen oder können, ist das eine Größenordnung mit der wir leben müssen.

Das Beispiel zeigt uns aber, daß wir keine vernünftige untere Schranke für den Aufwand eines *jeden* Algorithmus finden können, der unsere spezielle Kommunikation mit Schlüssel $s_0$ knackt: Der Aufwand für das Ausprobieren von $s_0$ ist schließlich vernachlässigbar klein. Wir müssen uns daher damit begnügen, Schranken für solche Algorithmen zu finden, die eine als gefährlich erachtete Erfolgswahrscheinlichkeit haben. Dabei können wir zwar $2^{-128}$ als relativ ungefährlich betrachten, aber $10^{-6}$ wäre beispielsweise im Bankenbereich bereits völlig inakzeptabel.

Wo genau wir die Grenze festlegen ist leider eine völlig akademische Frage, denn beim heutigen Stand der Mathematik haben wir nicht die geringste Chance, einen Mindestaufwand für irgendeine solche Grenze zu beweisen. In der Informatik beschäftigt sich die Komplexitätstheorie zwar seit Jahrzehnten auch mit solchen Fragen, aber die einzigen nichttrivialen Ergebnisse, die es bislang gab, beziehen sich auf untere Schranken unter der Nebenbedingung, daß gewisse Rechen- oder Logik-Operationen verboten sind – Bedingungen, auf die sich sicherlich keiner unserer Gegner einlassen wird. Komplexitätstheorie in ihrer heutigen Form ist daher für seriöse Kryptologie leider völlig irrelevant.

Damit haben wir also derzeit nicht einmal eine Chance, uns einigermaßen sicher (d.h. mit einer beweisbaren Wahrscheinlichkeit) gegen einen Gegner mit bekannter begrenzter Rechenkraft zu schützen, und wir müssen unsere Anforderungen noch einmal ganz deutlich absenken.

Dies geschieht dadurch, daß wir nun nicht nur annehmen, daß wir seine Rechenkraft einigermaßen abschätzen können, sondern wir nehmen zusätzlich auch noch an, daß die Algorithmen mit denen er arbeitet *im wesentlichen* – das heißt bis auf kleinere Verbesserungen – die allgemein bekannten sind. Wir schätzen also ab, wie lange wir selbst (geeignete,



den Möglichkeiten der erwarteten Gegner entsprechende Hardware vorausgesetzt) brauchen würden, um unser System zu knacken, und dividieren das Ergebnis durch einen hinreichend großen Sicherheitsfaktor.

Dieses neue Paradigma hat wesentliche Konsequenzen für die Auswahl unserer Kryptoverfahren: Bei einem neuen, noch kaum untersuchten Verfahren, ist die Wahrscheinlichkeit groß, daß es in der ersten Zeit häufig neue Ideen gibt, um die Kryptanalyse dramatisch zu beschleunigen. Man muß damit rechnen, daß nicht alle diese Ideen öffentlich bekannt werden.

Von daher ist es sicherer, ältere und seit langer Zeit auch in der offenen Literatur untersuchte Verfahren zu verwenden. Natürlich kann auch da niemand ausschließen, daß immer wieder neue Ideen auftauchen – die Geschichte der Mathematik ist voll von Beispielen, daß Probleme nach Jahrzehnten oder gar Jahrhunderten plötzlich in einem neuen Licht betrachtet und gelöst wurden. Bei einem Problem, das intensiv in der offenen wissenschaftlichen Literatur diskutiert wird, ist es allerdings unwahrscheinlich (wenn auch keinesfalls unmöglich – die Sicherheit haben wir nur mit dem *one time pad* und ähnlichen Verfahren), daß wirklich wesentliche neue Erkenntnisse lange Zeit aus der offenen Literatur ferngehalten werden können: Doppelentdeckungen sind in der heutigen Wissenschaft erstaunlich häufig.

Abgesehen vom Höchstsicherheitsbereich beruht die gesamte heutige Kryptographie auf diesem eher pragmatischen Sicherheitsmodell. Zumindest bislang ist man damit recht gut gefahren: Zwar gab es spektakuläre und erfolgreiche Angriffe auf Spezialfälle häufig angewandter allgemein akzeptierter Verschlüsselungsalgorithmen; am bekanntesten wurden wohl der auf den einfachen DES mittels DES-Cracker durch die Electronic Frontier Foundation, der von SERGE HUMPICHS auf die ersten Chipkarten, die in Frankreich anstelle von EC-Karten verwendet werden, und auch der Angriff einer Gruppe französischer Studenten auf die Verschlüsselung einer amerikanischen Exportversion von Netscape.

In allen diesen Fällen waren aber die Parameter so gewählt, daß die Verschlüsselung mit wohlbekannten Verfahren und vertretbarem Aufwand geknackt werden konnte, ohne daß dies irgendeinen Experten überrascht

hätte: Schon Jahre vor den entsprechenden Angriffen wurde immer wieder auf die unzulängliche Sicherheit von Systemen mit Parametern der verwendeten Größenordnung hingewiesen.

Leider zeigt sich hier eines der Grundprobleme der praktischen Kryptologie: Da es immer einen gewissen Aufwand erfordert, ein neues System einzuführen oder bei einem alten System die Schlüssellänge zu erhöhen, schrecken viele Anwender aus Kostengründen davor zurück, ihr System regelmäßig aufzurüsten. Hacker und Kryptanalytiker, die sich meist deutlich besser in der Kryptologie auskennen, bemühen sich dagegen, stets auf dem neuesten Stand zu sein. Auf diese Weise wird fast jedes ursprünglich sichere Kryptosystem im Laufe der Jahre zwangsläufig unsicher.

## §3: Wie wählt man zufällige Schlüssel?

Alle bislang diskutierten Verfahren verlieren deutlich an Sicherheit, sobald ihre Schlüssel nicht mehr zufällig gewählt werden: Selbst der *one time pad* wird völlig unsicher, wenn wir als Schlüssel deutschen Klartext an Stelle einer Zufallsfolge nehmen, denn auf Grund der Redundanz der deutschen Sprache in einer Größenordnung von rund 75% hat diese Summe immer noch etwa 50% Redundanz, was nicht schwer zu knacken ist.

### a) Was ist Zufall?

Im Alltagsgebrauch bezeichnen wir ein Ereignis als zufällig, wenn es keine Erklärung gibt, warum ausgerechnet dieses Ereignis an Stelle von mehreren ebenfalls möglichen Alternativen eingetreten ist: Fällt beim Würfel die Zahl „drei“, so ist das Zufall.

Untersuchen wir allerdings die Bewegung des Würfels genauer, so können wir ohne größere Probleme aus den Anfangsbedingungen (Ort, Geschwindigkeit, Drehimpuls des Würfels beim Abwurf) zumindest im Prinzip dessen Bahn berechnen und das Ergebnis vorhersagen; es ist also nicht mehr zufällig. In der Tat gibt es Spieler, die in der Lage sind, mit recht guter Trefferwahrscheinlichkeit jede gewünschte Zahl zu



würfeln. Bei genauerer Betrachtung des Vorgangs wir die Zufälligkeit des Ergebnisses hier also doch eher problematisch.

In der Tat ist es algorithmisch unmöglich, zu entscheiden, ob eine gegebene Zahlenfolge zufällig ist; wir können nie ausschließen, daß wir einfach das korrekte Bildungsgesetz noch nicht gefunden haben.

### b) Physikalische Zufallsquellen

Auch bei physikalischen Phänomenen, die uns zufällig erscheinen, müssen wir immer die Möglichkeit im Auge behalten, daß wir vielleicht einfach noch nicht die korrekte Theorie zu ihrer Erklärung gefunden haben. Zumindest in der Quantentheorie gibt es allerdings durchaus Sätze, wonach das Verhalten gewisser Systeme *nicht* mit bislang noch unbekannten „verborgenen Parametern“ deterministisch erklärt werden kann, und Phänomene wie der radioaktive Zerfall oder thermisches Rauschen liefern Werte, die als zufällig interpretiert werden können. Sie sind zwar im allgemeinen nicht gleichverteilt, aber dieses Problem läßt sich durch geeignete statistische Aufbereitung beheben.

Dem Normalanwender stehen physikalische Quellen jedoch üblicherweise nicht zur Verfügung. Trotzdem gibt es auch auf seinem Computer eine ganze Reihe von nicht vorhersehbaren Ereignissen, die keineswegs alle auf Softwarefehlern beruhen: Mißt man etwa den Abstand zwischen zwei Tastaturinterrupts mit einer Genauigkeit von einer Tausendstel Sekunde, so verhält sich die letzte Ziffer sicherlich zufällig: Niemand kann seine Bewegungen mit einer Genauigkeit in diesem Bereich steuern. Auch aus Mausbewegungen, Festplattenzugriffen und aus Netzwerkaktivitäten lassen sich entsprechende Zufallszahlen gewinnen.

Linux (und verschiedenen andere UNIX-Systeme) sammeln die so gewonnene Entropie und stellen sie in /dev/random als Folge gleichverteilter Zufallszahlen zur Verfügung; mit

```
dd if=/dev/random of=Dateiname bs=1 count=n
```

lassen sich also $n$ Zufallsbytes gewinnen – sofern hinreichend viel Entropie vorhanden ist. Andernfalls blockiert /dev/random und liefert erst dann wieder neue Bytes, wenn neue zufällige Ereignisse eingetreten sind.

Eine zweite Spezialdatei, /dev/urandom, greift ebenfalls auf den Entropiepool des Betriebssystems zu, liefert aber bei nicht ausreichender Entropie einfach Zufallsbytes in reduzierter Qualität ohne zu blockieren.

Unter cygwin stehen /dev/random und /dev/urandom im Prinzip auch für Windows-Anwender zur Verfügung, allerdings liefern dann beide nur Pseudozufallszahlen in üblicher Windowsqualität, die auf keinen Fall für kryptographische Zwecke verwendet werden dürfen.

### c) Pseudozufallszahlen

Rechnungen eines Computers sollten strikt deterministisch sein und können daher ihrer Natur nach keine wirklich zufälligen Phänomene liefern. Trotzdem lassen sich Folgen erzeugen, die sich bezüglich vieler Aspekte wie „echte“ Zufallszahlen verhalten. Man bezeichnet solche Zahlen als Pseudozufallszahlen.

Was eine Folge von Pseudozufallszahlen wirklich sein soll, hat aus gutem Grund noch niemand mathematisch exakt definiert: Da reproduzierbarer Zufall ein Widerspruch in sich ist, kann es keine solche Definition geben. Ob eine gegebene deterministisch berechnete Folge von Zahlen sich (pseudo)zufällig verhält, kann nur anhand ihrer Nützlichkeit für eine vorgegebene Anwendung entschieden werden.

Die *heuristische* und alles andere als exakte Definition einer Folge von Pseudozufallszahlen ist daher, daß sich diese Zahlen für die jeweils anstehende Anwendung so verhalten, daß man sie *praktisch* nicht von einer echt zufälligen Zahlenfolge unterscheiden kann.

Diese Abhängigkeit von der Anwendung hat durchaus auch praktische Bedeutung: Falls man eine jener Folgen von Pseudozufallszahlen, die für Simulationen populär sind, in der Kryptographie verwendet, kann man praktisch sicher sein, daß das so manipulierte Kryptoverfahren keinerlei Sicherheit mehr bietet. Verwendet man umgekehrt kryptographisch geeignete Pseudozufallszahlen für eine Simulation, so führt dies wegen der erheblich größeren Komplexität von deren Erzeugung zu einem nicht mehr vertretbaren Rechenaufwand.



Für Simulationen, die nicht mehr als einige Tausend Zufallszahlen brauchen, sind die sogenannten linearen Kongruenzgeneratoren die populärste Methode; darauf beruhen auch zahlreiche Standardprogramme zur Erzeugung von Pseudozufallszahlen. Wir wollen uns anhand dieses Beispiels überlegen, welche Probleme es bei Pseudozufallszahlen gibt, und warum speziell dieser Generator kryptographisch *keinerlei* Sicherheit bietet.

Ein linearer Kongruenzgenerator hängt ab von drei Parametern $a, N \in \mathbb{N}$ und $b \in \mathbb{N}_0$, wobei oftmals $b = 0$ gesetzt wird, sowie von einem Anfangswert $x_0$. Die weiteren Werte werden berechnet auf Grund der Vorschrift

$$x_{n+1} = ax_n + b \bmod N\,.$$

(Falls man Werte zwischen Null und Eins möchte, nimmt man einfach die Zahlen $x_n/N$ oder, wenn beide Ränder vorkommen sollen, $x_n/(N-1)$.)

Daß eine derart erzeugte Folge nicht wirklich zufällig sein kann, ist klar: Es gibt nur endlich viele Zahlen modulo $N$, und sobald $x_n = x_m$ ist, muß auf Grund der Konstruktionsvorschrift auch $x_{n+k} = x_{m+k}$ sein für jede natürliche Zahl $k \in \mathbb{N}$, die Folge muß also periodisch werden.

Diese Eigenschaft teilt sie allerdings mit *jeder* Zahlenfolge, die ein Computer ohne Zugriff auf externe Informationen erzeugen kann: Als endliche Maschine hat ein Computer nur endlich viele Zustände, so daß bei jeder Iteration irgendwann ein Zustand erreicht wird, der schon einmal da war. Da bei einer deterministischen Rechenvorschrift der jeweilige Folgezustand eindeutig durch den augenblicklichen Zustand bestimmt ist, muß das weitere Verhalten somit periodisch sein.

Folgen von Pseudozufallszahlen sind daher stets periodisch; das ist so lange kein Problem, wie die Periode größer ist als die Anzahl benötigter Zufallszahlen. Ein entscheidendes Kriterium bei der Beurteilung eines Pseudozufallsgenerator ist also dessen Periode.

Beim linearen Kongruenzgenerator wollen wir diese nur im (in der Praxis der häufigen) Fall $b = 0$ betrachten, da hier die dazu notwendige Mathematik etwas einfacher ist.

Wir betrachten also die Iterationsvorschrift $x_{n+1} = ax_n \bmod N$, die sich leicht umschreiben läßt in die Formel $x_n = a^n x_0 \bmod N$, oder allgemeiner auch $x_n = a^k x_{n-k}$, was bei der Arbeit mit Parallelrechnern oft nützlich ist, da man so die Berechnung der Pseudozufallszahlen auf mehrere Prozessoren verteilen kann.

Als weitere Vereinfachung beschränken wir uns auf Primzahlen $N = p$; zum Glück sind diese bei linearen Kongruenzgeneratoren in der Tat mit die populärsten Moduln. (Ihre schärfsten Konkurrenten sind, aus nahe liegenden Gründen, die Zweierpotenzen.)

Für Primzahlen sagt uns der kleine Satz von FERMAT, daß

$$a^p \equiv a \bmod p \quad \text{für alle } a \in \mathbb{Z}$$

(Zur Erinnerung: Das ist eine einfache Folgerung aus der binomischen Formel

$$(x+y)^p = \sum_{k=0}^{p} \binom{p}{k} x^k y^{p-k} \quad \text{mit} \quad \binom{p}{k} = \frac{p(p-1)\cdots(p-k+1)}{k!} .$$

Für $k \neq 0, p$ hat nämlich das $p$ im Zähler des Binomialkoeffizienten keine Entsprechung im Nenner, $\binom{p}{k}$ ist also durch $p$ teilbar und somit ist $(x+y)^p \equiv x^p + y^p \bmod p$. Wendet man dies im Spezialfall $y = 1$ induktiv an, folgt der kleine Satz von FERMAT.)

Damit ist klar, daß ein linearer Kongruenzgenerator mit Modul $p$ höchstens die Periode $p-1$ haben kann. Die muß er allerdings nicht haben, denn es gibt auch Elemente, für die bereits eine kleinere als die $(p-1)$-te Potenz Eins ist: Für $p = 13$ beispielsweise ist nur bei $a = 2, 6, 7, 12$ erst die zwölfte Potenz gleich Eins; bei $a = 4$ und $a = 10$ ist es bereits die sechste, bei $a = 5, 8$ die vierte, bei $a = 3, 9$ die dritte, bei $a = 12$ die zweite, und bei $a = 1$ natürlich die erste.

Allgemein gilt, daß der kleinste Exponent, für den $a^n \equiv 1 \bmod p$ ist stets Teiler von $p-1$ ist, und daß es stets Elemente $a$ gibt, für die er *gleich* $p-1$ ist.

Der Beweis ist nicht sonderlich schwer: Sei $d$ der größte gemeinsame Teiler von $n$ und $p-1$. Nach dem erweiterten EUKLIDischer Algorithmus) (mit dem wir uns später noch genauer beschäftigen werden), läßt sich $d$ als ganzzahlige Linearkombination

$$d = rn + s(p-1)$$



von $n$ und $p-1$ schreiben. Da $a^n$ und (nach dem kleinen Satz von FERMAT) $a^{p-1}$ beide modulo $p$ gleich Eins sind, ist somit auch

$$a^d = a^{rn+s(p-q)} = (a^n)^r \cdot (a^{p-1})^2 = 1 \cdot 1 = 1 .$$

Da $n$ die kleinste Zahl mit dieser Eigenschaft ist, muß $n \leq d$ sein; andererseits ist $d$ als ggT von $n$ und $p-1$ ein Teiler von $n$, also muß $n = d$ sein, d.h. $n$ teilt $p-1$.

Um zu zeigen, daß es Elemente gibt, für die $n = p-1$ ist, zerlegen wir $p-1$ in seine Primfaktoren:

$$p-1 = q_1^{e_1} \cdots q_m^{e_m}$$

mit Primzahlen $q_1, \ldots, q_m$. Für jedes dieser Primzahlen $q_j$ gibt es höchstens $(p-1)/q_j$ Zahlen $a \bmod p$, für die $a^{(p-1)/q_j} = 1$ ist, denn dies sind die Nullstellen des Polynoms $X^{(p-1)/q_j} - 1$ im Körper mit $p$ Elementen, und ein Polynom kann in einem Körper höchstens so viele Nullstellen haben, wie sein Grad angibt. Also gibt es zu jeder Primzahl $q_j$ mindestens eine Zahl $a_j$, für die $a_j^{(p-1)/q_j} \neq 1$ ist. Dazu bilden wir die $(p-1)/q_j^{e_j}$-te Potenz

$$b_j = a_j^{(p-1)/q_j^{e_j}} .$$

Für diese ist

$$b_j^{q_j^{e_j}} = \left(a_j^{(p-1)/q_j^{e_j}}\right)^{q_j^{e_j}} = a_j^{(p-1)/q_j^{e_j}} \cdot q_j^{e_j} = a_j^{p-1} \equiv 1 \bmod p ,$$

aber

$$b_j^{q_j^{e_j-1}} = \left(a_j^{(p-1)/q_j^{e_j}}\right)^{q_j^{e_j-1}} = a_j^{(p-1)/q_j^{e_j}} \cdot q_j^{e_j-1} = a_j^{(p-1)/q_j} \not\equiv 1 \bmod p .$$

Das Produkt $a$ dieser Zahlen $b_j$ hat nun die gewünschte Eigenschaft: Gäbe es nämlich eine Zahl $n < p-1$ mit $a^n \equiv 1 \bmod p$, so müßte $n$ mindestens eine der Zahlen $(p-1)/q_j$ teilen, so daß für mindestens ein $j$ auch $a^{(p-1)/q_j} \equiv 1 \bmod p$ wäre. Aber

$$a^{(p-1)/q_j} = b_1^{(p-1)/q_j} b_2^{(p-1)/q_j} \cdot b_m^{(p-1)/q_j} \equiv b_j^{(p-1)/q_j} \not\equiv 1 \bmod p ,$$

denn für $k \neq j$ ist $(p-1)/q_j$ ein Vielfaches von $q_k^{e_k}$, so daß $b_k^{(p-1)/q_j} \equiv 1 \bmod p$ ist.

Zahlen, für die $p-1$ der kleinste Exponent $n$ ist mit der Eigenschaft, daß $a^n \equiv 1 \bmod p$, heißen *primitive Wurzeln modulo p*; für einen guten linearen Kongruenzgeneratoren modulo $p$ sollte also der Multiplikator $a$ eine primitive Wurzel modulo $p$ sein.

Es gibt keine geschlossene Formel, mit der man sich für eine gegebene Primzahl $p$ eine primitive Wurzel modulo $p$ verschaffen kann; man findet primitive Wurzeln aber trotzdem selbst modulo sehr großer Primzahlen,

wie man sie in der Kryptographie benötigt, recht schnell, indem man zufällige Werte für $a$ wählt und testet, ob deren Ordnung kleiner ist als $p-1$.

Mit einer primitiven Wurzel haben wir allerdings bei weitem noch keine Garantie für einen guten Pseudozufallszahlengenerator: Wie eingangs erwähnt, soll er sich schließlich bezüglich der jeweils interessierenden Anwendung möglichst zufällig verhalten.

Eine häufige Anwendung besteht darin, zufällige Punkte in einem höherdimensionalen Raum zu erzeugen. Das ist natürlich kein Problem: Im Zweidimensionalen etwa nehmen wir zu einer Folge $x_0, x_1, \ldots$ von Zufallszahlen einfach die Folge der Punkte

$$(x_0, x_1),\ (x_2, x_3),\ (x_4, x_5),\ , \ldots\ .$$

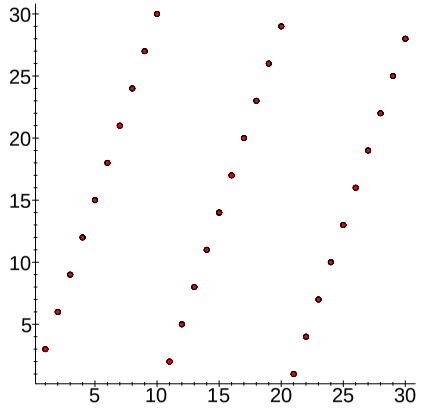


Abb. 3: Die Punkte $(x_n, x_{n+1})$ für $a = 3$

Als überschaubares Beispiel betrachten wir den (für jede praktische Anwendung natürlich viel zu kleinen) Fall $p = 31$. Die kleinste primitive Wurzel modulo 31 ist drei, also betrachten wir die Folge

$$x_{n+1} = 3x_n \bmod 31\ .$$



Mit $x_0 = 15$ etwa erhalten wir so die Folge

$$15, 14, 11, 2, 6, 18, 23, 7, 21, 1, 3, 9, 27, 19, 26, 16, 17,$$
$$20, 29, 25, 13, 8, 24, 10, 30, 28, 22, 4, 12, 5, 15, \ldots\ ,$$

die in der Tat gut durcheinander geschüttelt aussieht. Betrachten wir aber die dreißig möglichen Paare $(x_n, x_{n+1})$, so zeigt Abbildung vier, daß hier wahrlich kein Grund zur Begeisterung besteht: Alle Punkte liegen auf nur drei Geraden! In einer kryptographischen Anwendung würde es diese Tatsache dem Gegner sehr einfach machen, anhand der Steigung dieser Geraden auf den Multiplikator $a$ zu schließen.

Wählen wir stattdessen etwa $a = 11$, so erhalten wir eine deutlich bessere zweidimensionale Darstellung:

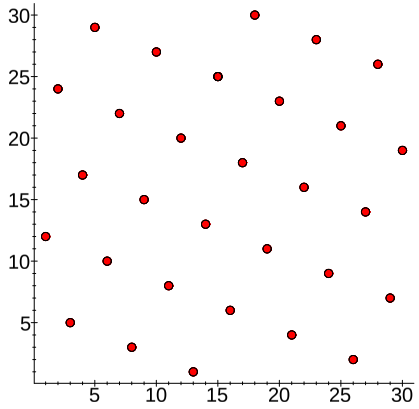


Abb. 4: Die Punkte $(x_n, x_{n+1})$ für $a = 11$

An der bloßen Folge der Zahlenwerte war das nicht zu erkennen: Wenn wir wieder mit $x_0 = 15$ starten, erhalten wir die Folge

$$15, 25, 21, 4, 17, 18, 30, 19, 11, 8, 3, 5, 29, 7, 22, 16, 6,$$
$$10, 27, 14, 13, 1, 12, 20, 23, 28, 26, 2, 24, 9, 15, \ldots\ ,$$

die weder besser noch schlechter als die obige aussieht.

Solche Probleme treten beileibe nicht nur bei trivialen Spielzeugbeispielen wie dem obigen auf: In der Zeit der Großrechner war der am weitesten verbreitete Pseudozufallsgenerator ein Algorithmus namens `RANDU`, der im wesentlichen äquivalent war zum folgenden linearen Kongruenzgenerator:

$$x_{n+1} = (2^{16} + 3)x_n \bmod 2^{31} .$$

Seine Popularität war sicherlich nicht zuletzt darauf gegründet, daß er sich wegen der beiden Zweierpotenzen sehr effizient implementieren läßt; in der Tat nutzt das Original auch noch das Überlaufbit aus, so daß der Algorithmus mit 32-Bit-Worten und der üblichen 32-Bit-Arithmetik auskommt. Angeblich soll er auch heute noch in einigen Unterprogrammbibliotheken zu finden sein.

Bei ein- oder zweidimensionale Simulationen gibt er keinerlei Anlaß zum Klagen, und auch im Dreidimensionalen sehen die Bilder, die er erzeugt, auf den ersten Blick sehr gut aus:

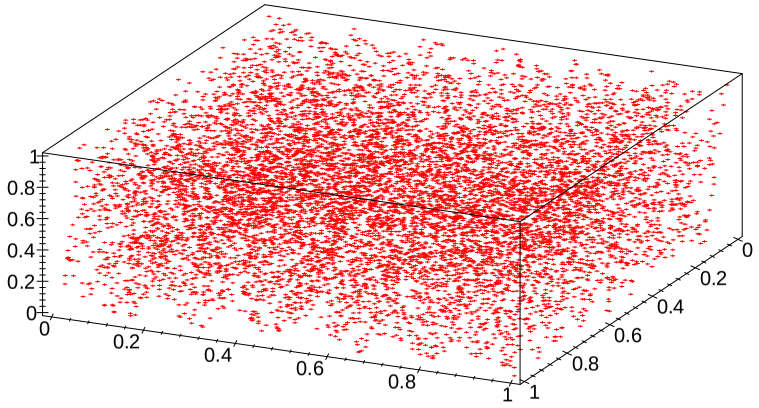


Abb. 5: 10000 von RANDU erzeugte Punkte im Dreidimensionalen

(Der Übersichtlichkeit halber sind in diesem und dem folgenden Bild alle Koordinaten durch $2^{31}$ dividiert.)

Ändert man allerdings den Blickwinkel, so sieht man, daß tatsächlich alle Punkte auf nur fünfzehn Ebenen liegen:



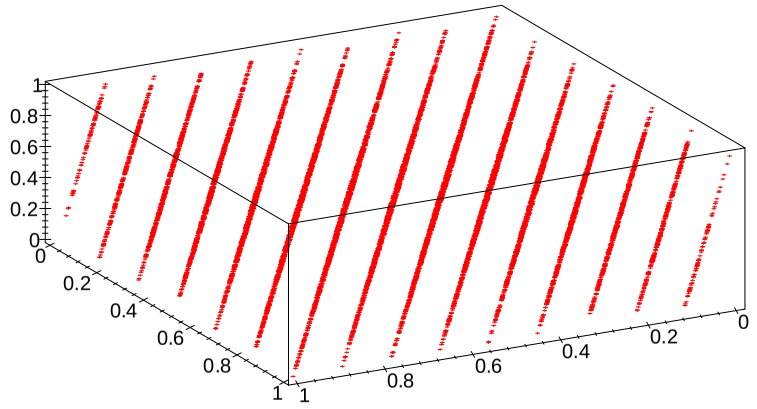


Abb. 6: Dasselbe unter einem anderen Blickwinkel

Das läßt sich auch leicht theoretisch verstehen: Nach Definition ist für jedes $n$ modulo $2^{31}$

$$\begin{aligned} x_{n+2} &\equiv (2^{16} + 3)^2 x_n = (2^3 2 + 6 \cdot 2^{16} + 9)x_n \\ &\equiv (6 \cdot 2^{16} + 9)x_n = \big(6(2^{16} + 3) - 9\big)x_n \\ &= 6(2^{16} + 3)x_n - 9x_n \equiv 6x_{n+1} - 9x_n \bmod 2^{31} . \end{aligned}$$

Damit ist

$$x_{n+2} - 6x_{n+1} + 9x_n \equiv 0 \bmod 31 \qquad \text{für alle } n ;$$

für jedes $n$ gibt es also ein $k \in \mathbb{Z}$, so daß gilt

$$x_{n+2} - 6x_{n+1} + 9x_n = k \cdot 2^{31} .$$

Der kleinstmögliche Wert von $k$ wird angenommen, wenn $x_{n+1}$ möglichst groß ist, also $2^{31} - 1$ oder knapp darunter, $x_n$ und $x_{n+2}$ aber sehr klein. Da sich $-6 \cdot 2^{31}$ offensichtlich nicht erreichen läßt, ist der minimal mögliche Wert von $k$ gleich $-5$.

Für den größtmöglichen Wert von $k$ muß $x_{n+1}$ nahe bei der Null liegen und $x_{n+2}$ sowie $x_n$ nahe $2^{31} - 1$; der Extremfall $10 \cdot 2^{31}$ wird dabei offenbar stets unterschritten, das maximal mögliche $k$ ist also neun.

Somit kann $k$ nur die fünfzehn Werte zwischen $-5$ und 9 annehmen, so daß alle Punkte auf nur fünfzehn Ebenen liegen.

Wie man sieht, kann beim Design eines Zufallsgenerators also vieles schief gehen.

Selbst wenn alles gut geht, gibt es allerdings für die Kryptographie die entscheidende Schwäche, daß die Entropie einer Pseudozufallsfolge nicht größer sein kann als die Entropie der Parameter des Algorithmus: Sobald man diese Information hat, läßt sich schließlich die gesamte Folge rekonstruieren.

Gerade der lineare Kongruenzgenerator verhält sich hier sehr schlecht: Falls man mit der Standardarithmetik eines PCs rechnet, müssen die Parameter relativ klein sein und lassen sich darüber hinaus bereits auf der Grundlage weniger Folgenglieder einfach via Lineare Algebra über $\mathbb{F}_p$ berechnen. Wählt man also einen solchen Generator beispielsweise als Grundlage für einen *one time pad*, so läßt sich bereits mit wenigen Bytes bekanntem Klartext der gesamte Text entschlüsseln.

Es gibt auch krypographisch stärkere Generatoren, bei denen die Berechnung der Parameter aus einer Zufallsfolge nach heutigem Kenntnisstand (in der offenen Literatur) rechnerisch nicht mit vertretbarem Aufwand möglich ist. Gegenüber dem BAYESschen Gegner sind selbstverständlich auch solche Generatoren nicht sicher, aber das gilt auch für die meisten anderen Verfahren, die wir in diesem Semester noch betrachten werden.

### d) Test von Zufallszahlen

Da es nicht möglich ist, algorithmisch zu erkennen, ob eine gegebene Folge zufällig ist oder nicht, kann natürlich kein Test garantieren, daß eine gegebene Folge zufällig ist: Er kann höchstens erkennen, daß sie sich unter irgendeinem Aspekt *nicht* zufällig verhält.

Dazu stellt die Mathematik eine ganze Reihe von Verfahren zur Verfügung. Hier sollen nur drei der Möglichkeiten erwähnt werden:

Die Entropie einer Zufallsfolge kann natürlich nicht größer sein als der Parameter des Generatorss; die Entropie pro Zeichen geht also asymptotisch gegen Null. Betrachtet man allerdings die Folge nur unter dem



Gesichtspunkt von Häufigkeiten oder Kontakthäufigkeiten (und gegebenenfalls auch noch MARKOV-Ketten höherer Ordnung), wird man im allgemeinen eine deutlich höhere Entropie pro Zeichen erhalten. Je größer diese ist, desto mehr Vertrauen wird man dem Generator entgegenbringen.

Als zweites kann man testen, ob die Folgenglieder einigermaßen gleichverteilt sind. Dazu dient ein Standardverfahren der Statistik, der sogenannte $\chi^2$-Test: Für eine Folge $x_1, \ldots, x_N$ mit $x_i$ aus einem Intervall $[a, b]$ unterteilt man das Intervall für geeignete Zahlen $r$ in $r$ gleich große Teilintervalle der Länge $(b-a)/r$; die Anzahl der $x_i$ im $j$-ten Teilintervall sei $n_j$.

Bei optimaler Gleichverteilung sollten in jedem dieser Teilintervalle ungefähr $N/r$ Zahlen liegen. Die Größe

$$\chi^2 = \sum_{j=1}^{r} \frac{\left(a_j - N/r\right)^2}{N/r}$$

ist also ein Maß der Inhomogenität. Wie die Statistik zeigt, genügt sie einer sogenannten $\chi^2$-Verteilung mit $r-1$ Freiheitsgraden. (Die $-1$ kommt daher, daß $a_r = N - a_1 - \cdots - a_{r-1}$ durch die restlichen $a_j$ festgelegt ist.)

In Tabellenwerken kann man nachschauen, mit welcher Wahrscheinlichkeit $\chi^2$ für gegebenes $r$ einen Wert annimmt, der kleiner oder größer als eine vorgegebene Schranke ist. Zu kleine Werte deuten auf eine wenig zufällige Folge hin, zu große auf mangelnde Homogenität.

Ein dritter Test schließlich soll nur kurz erwähnt werden: Betrachtet man die Summe mehrerer periodischer Funktionen zu verschiedenen, idealerweise inkommunserablen Perioden, und tastet diese an äquidistanten Stellen ab, so sieht das Ergebnis recht zufällig aus und besteht auch die bisherigen Tests. Trotzdem würden wir eine solche Folge nicht als wirklich zufällig bezeichnen.

Mit dem sogenannten Spektraltest erkennt man die Perioden, indem man einfach die diskrete FOURIER-Transformation der Folge betrachtet: Falls es hier deutlich ausgeprägte Frequenzspitzen gibt, wird man die Folge als nicht hinreichend zufällig ablehnen.

## §4: Literatur

Eine sehr ausführliche Darstellung der statistischen und informationstheoretischen Ansätze findet man in

ALAN G. KONHEIM: *Cryptography: a primer,* Wiley, New York, 1981

Ein neueres Buch, in dem diese Ansätze eine große Rolle spielen, und das auch auf die Erzeugung gleichverteilter Zufallszahlen auf Grundlage physikalischer Prozesse eingeht, ist

DANIEL NEUENSCHWANDER: *Probabilistic and statistical methods in cryptology,* Springer Lecture Notes in Computer Science **3028**, 2004

Die Herleitung der Entropiedefinition in diesem Kapitel folgte

A.I. KHINCHIN: *Mathematical foundations of information theory,* Dover, 1957,

wo auch zahlreiche weitere Aspekte diskutiert werden. Um die Anwendung des BAYESschen Ansatzes im Alltagsleben geht es in

LUC BOVENS, STEPHAN HARTMANN: *Bayesian epistemology,* Oxford University Press, 2003

Ein universelles Lehrbuch der Kryptologie mit einem Kapitel über diese Ansätze ist das bereits zitierte Buch

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

Ausführlichere Informationen über Erzeugung und Test von Zufallszahlen für Simulationen findet man beispielsweise in

DONALD E. KNUTH: *The art of computer programming, vol. 2: Seminumerical algorithms,* Addison Wesley, $^2$1981

und in erheblich größerem Umfang in

JAMES E. GENTLE: *Random number generation and Monte Carlo methods,* Springer, 1998

# Kapitel 3
# Blockchiffren und ihre Kryptanalyse I: DES

## §1: Grundlagen

Bei den sogenannten klassischen oder symmetrischen Kryptosystemen gibt es einen zwischen Sender und Empfänger vereinbarten Schlüssel, der sowohl zur Ver- als auch zur Entschlüsselung benutzt wird. Je nach Vorgehensweise unterscheidet man zwei Arten von Chiffren:

- Stromchiffren *und*
- Blockchiffren.

Bei *Stromchiffren* wird ein Text oder sonstiger Datenstrom buchstaben- oder byteweise oder in welchen Einheiten auch immer die Daten ankommen chiffriert, wobei der Schlüssel natürlich von Buchstabe zu Buchstabe wechselt. *Blockchiffren* fassen die Daten zu Blöcken einer festen Länge zusammen und verschlüsseln diese Blöcke als ganzes.

Stromchiffren werden häufig angewandt zur Übermittlung von Satellitendaten, Satellitenfernsehen und ähnlichen kontinuierlichen Datenströmen; Blockchiffren haben ihre Bedeutung im Bankenbereich, sowohl im elektronischen Zahlungsverkehr wie auch zur Identifikation der Benutzer an Geldautomaten, zur Sicherung der Kommunikation zwischen Computern und in ähnlichen Anwendungen.

### a) Die Sicherheit einer Blockchiffre

Typischerweise faßt eine Blockchiffre die zu übermittelnden Daten zusammen zu Blöcken von 64 Bit (bei alten Systemen) oder 128 Bit (bei neueren). Jeder Block wird in derselben (von einem Schlüssel abhängigen) Weise durch einen anderen Block derselben Länge ersetzt; ist $\mathcal{B}$

die Menge aller Blöcke der vorgegebenen Länge, so besteht das Kryptosystem also aus bijektiven Abbildungen

$$T_s : B \to B; \qquad s \in S\,.$$

Jedes $T_s$ ist damit ein Element der symmetrischen Gruppe $S_{\#B}$, im Gegensatz zu den monoalphabetischen Buchstabensubstitutionen können wir allerdings nicht mehr alle Permutation zulassen: Falls etwa $B$ aus Blöcken der Länge 64 Bit besteht, hat $B$ immerhin schon $2^{64}$ Elemente, es gibt also $2^{64}!$ Permutationen. Nach STIRLING ist in erster Näherung

$$\log_2(2^{64}!) \approx 2^{64} \log_2 2^{64} = 64 \cdot 2^{64} = 2^{70}\,,$$

zur Festlegung einer Permutation braucht man also ungefähr $2^{70}$ Bit oder $2^{67}$ Byte oder $2^{57}$ kB oder $2^{47}$ MB oder $2^{37}$ GB oder $2^{27}$ TB oder ..., jedenfalls viel zu viel.

Aus diesem Grund muß die Abbildungsvorschrift konkreter definiert werden; allerdings sollten die Substitutionen wenn möglich immer noch wie zufallsverteilt in der symmetrischen Gruppe liegen. Beispielsweise sollten sie keine Untergruppe bilden, denn die Kenntnis von Erzeugenden und Relationen dieser Untergruppe könnte einem Gegner Ansatzpunkte zur Entschlüsselung geben.

Ansonsten hängt die Sicherheit einer jeden Blockchiffre stark vom Inhalt der Klartexte und der Schlüssellänge ab: Für deutschen Klartext mit ASCII-Kodierung der Buchstaben etwa enthält ein Block von 64 Bit acht Buchstaben; geht man von einer Redundanz der deutschen Sprache in der Größenordnung von etwa 70% aus, enthält er also eine Information von

$$8 \times 0{,}3 \times \log_2 26 \approx 11{,}28\,\text{Bit}\,;$$

tatsächlich übermittelt werden 64 Bit Chiffretext, so daß in erster Näherung ein Block etwa 52 Bit Schlüsselinformation liefert. Bei einer Schlüssellänge von 64 Bit reichen dem BAYESschen Gegner also im allgemeinen bereits zwei Blocke zur Dekodierung, bei einer Schlüssellänge von 128 Bit braucht er drei.

Sicherer ist ein System, das komprimierten Text übermittelt, da hier die Redundanz bereits deutlich reduziert ist; man muß allerdings beachten,



daß bei vielen gängigen Komprimierungsprogrammen eine ganze Reihe von Anfangsbytes auf feste Werte gesetzt werden, so daß der BAYESsche Gegner mit bekanntem Klartext arbeiten kann.

Für die praktischen Beurteilung einer Blockchiffre hat sich heutzutage folgendes Kriterium durchgesetzt:

- Die Schlüssellänge muß so groß sein, daß systematisches Durchprobieren für die zu erwartenden Gegner unrealistisch ist
- Das Verfahren muß so gewählt werden, daß es keine Attacke gibt, die nennenswert schneller ist als systematisches Durchprobieren

Beide Kriterien sind eher vage formuliert, aber wie wir in Kapitel 2, §2*g)* gesehen haben, haben wir leider keine Chance, konkretere Forderungen nachzuprüfen.

### b) Beispiel: Hill-Chiffren

Ein möglicher Ansatz für eine Blockchiffre wäre etwa, daß man die Substitution der Blöcke durch lineare Gleichungen angibt: Für $\mathbf{x} = (x_1, \ldots, x_N)$ sei die Verschlüsselung $\mathbf{y} = (y_1, \ldots, y_N)$ mit

$$y_i = \sum_{j=1}^{N} a_{ij} x_j + b_i, \qquad a_{ij}, b_j \in \mathbb{F}_2\,,$$

wobei $N$ natürlich die Blocklänge bezeichnet und $\mathbb{F}_2 = \{0, 1\}$ den Körper mit zwei Elementen mit Rechenoperationen modulo zwei.

Der Schlüssel hätte dann die Länge $N(N + 1)$, was handhabbar ist; man muß aber darauf achten, daß die Matrix $A = (a_{ij})$ invertierbar ist, da es sonst keine eindeutige Entschlüsselung gibt. Solche Chiffren heißen HILL-Chiffren nach LESTER HILL, der sie allerdings nicht über $\mathbb{F}_2^N$ betrachtete, sondern über $(\mathbb{Z}/26)^N$, wo sie eine Verallgemeinerung der CAESAR-Chiffren darstellen.

Unsicher sind sie sowohl über $\mathbb{F}_2$ als auch über $\mathbb{Z}/26$, sofern der Gegner über hinreichend viel korrespondierenden Klartext und Chiffretext verfügt und den GAUSS-Algorithmus zur Lösung linearer Gleichungssysteme kennt: Mit $N + 1$ linear unabhängigen Klartextblöcken kann er über einem Körper die Spalten von $A$ sowie auch $b$ berechnen; bei einem

Ring wie $\mathbb{Z}/26$ kann es noch Probleme mit gemeinsamen Teilern geben, die aber den Aufwand auch nicht wesentlich größer machen. Auch ohne Kenntnis von Klartext (die man in der Praxis nie ausschließen kann) gibt es Verfahren zur Kryptoanalyse.

### c) Diffusion und Konfusion

Nach CLAUDE SHANNON sollte ein Kryptosystem auf zwei Techniken beruhen: **Diffusion** und **Konfusion**.

*Diffusion* soll die Redundanz des Klartexts möglichst weiträumig über den Chiffretext verteilen. Dies könnte beispielsweise durch eine Permutation der Klartextbuchstaben erreicht werden (was die deutsche Armee im ersten Weltkrieg als hauptsächliches Kryptoverfahren benutzte); allerdings ist diese Permutation allein kryptographisch ziemlich unsicher: Da man beispielsweise weiß, daß auf ein „C“ in deutschen Klartext praktisch immer ein „H“ folgt oder auf ein „Q“ ein „U“, kann man bei hinreichend viel Chiffretext solche Permutationen leicht rückgängig machen. Bei guten Kryptoverfahren werden sie daher nur angewandt als eine von mehreren Komponenten. Diffusion wird auch erreicht, wenn man dafür sorgt, daß jedes Klartextbit möglichst viele Bit des Chiffretexts beeinflußt.

*Konfusion* soll dafür sorgen, daß der Zusammenhang zwischen Schlüssel und Chiffretext möglichst undurchsichtig ist; der Kryptanalytiker soll aus dem Chiffretext nur wenig Information über die Schlüsselverteilung gewinnen können. Der *one time pad* ist ein Beispiel dafür, wie Konfusion zu perfekter Sicherheit führen kann; die anderen Beispiele aus dem ersten Kapitel zeigen, daß reine Substitution bei kürzeren Schlüssellängen nicht sonderlich sicher ist. Wie das Beispiel der HILL-Chiffre zeigt, muß Konfusion insbesondere auch für Nichtlinearität sorgen; andernfalls reicht möglicherweise bereits die Lineare Algebra zur Kryptanalyse.

Moderne Kryptosysteme beruhen darauf, daß Diffusion und Konfusion *in Kombination* zu deutlich größerer Sicherheit führen als jede der beiden Strategien für sich allein.



## § 2: Feistelnetzwerke und der Aufbau des DES

### a) Feistelnetzwerke

Bis vor kurzem waren fast alle gebräuchlichen Blockchiffren sogenannte FEISTEL-Netzwerke, die in *Runden* arbeiten: In jeder Runde wird nur ein Teil des Blocks und ein Teil des Schlüssels verwendet, aber nachdem alle der (typischerweise sechzehn) Runden abgearbeitet sind, ist sichergestellt, daß die Ausgabe von allen Schlüsselbits und allen Klartextbits abhängt.

HORST FEISTEL entwickelte ab etwa 1960 bei IBM die ersten Blockchiffren, insbesondere auch das Lucifer-System, auf dessen Grundlage DES entwickelt wurde.

Sei $2N$ die Blocklänge des zu betrachtenden Systems. Dann gibt es eine Funktion

$$f\colon \mathbb{F}_2^k \times \mathbb{F}_2^N \to \mathbb{F}_2^N \ ,$$

die sogenannte FEISTEL-Funktion, die aus $k$ Schlüsselbits und $N$ Nachrichtenbits wieder $N$ Bits produziert. Diese wird folgendermaßen angewandt: Vor Beginn der $i$-ten Runde wird die Nachricht aufgeteilt in eine linke Hälfte $L$ und eine rechte Hälfte $R$. Die $i$-te Runde ersetzt den Block $(L, R)$ durch

$$\big(f(s_i, R) \oplus L, R\big) \ ,$$

wobei $\oplus$ die Addition im Vektorraum $\mathbb{F}_2^N$ bezeichnet und $s_i$ den Schlüssel der $i$-ten Runde. Nach jeder außer der letzten Runde werden anschließend die linke und rechte Hälfte miteinander vertauscht – ansonsten würde ja die rechte Hälfte den gesamten Algorithmus unverändert durchlaufen.

Die FEISTEL-Funktion dient somit zur *Konfusion;* die Auswahl der Schlüsselbits für die jeweiligen Runden sowie auch die Vertauschung von linker und rechter Seite in jeder Runde zur Diffusion.

### b) Aufbau des DES

Als Beispiel betrachten wir den 1977 in USA als Standard eingeführten DES (Data Encryption Standard), der unter anderem auch unserem EC-Kartensystem zugrunde liegt. Er arbeitet mit einem Schlüssel der Länge

56 Bit (der mit acht Prüfbits auf 64 Bit verlängert wird, indem man an je sieben Schlüsselbits ein Paritätsbit anhängt) auf Blöcken der Länge 64.

Zunächst werden die Datenwörter

$$(x_1, x_2, \ldots, x_{64})$$

einer Anfangspermutation unterzogen, d.h. das Wort wird ersetzt durch

$$(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(64)}) ,$$

wobei die Folge der Zahlen $\pi(1), \ldots, \pi(64)$ der folgenden Tabelle entnommen wird:

| | |
|---|---|
| 58 50 42 34 26 18 10 2 | 60 52 44 36 28 20 12 4 |
| 62 54 46 38 30 22 14 6 | 64 56 48 40 32 24 16 8 |
| 57 49 41 33 25 17 9 1 | 59 51 43 35 27 19 11 3 |
| 61 53 45 37 29 21 13 5 | 63 55 47 39 31 23 15 7 |

Das Wort geht also auf

$$(x_{58}, x_{50}, x_{42}, \ldots, x_{23}, x_{15}, x_7) .$$

Danach beginnen die sechzehn Runden.

Die Anfangspermutation hat keine kryptographische Funktion: Da sie nicht vom Schlüssel abhängt und allgemein bekannt ist, kann sie jeder Kryptanalytiker leicht rückgängig machen. Ihr Sinn bestand anscheinend in erster Linie darin, Software-Attacken zu erschweren, denn Permutation sind aufwendig zu programmieren. (Bei Hardware-Implementierungen sind Permutationen natürlich sehr einfach und schnell durch Leitungskreuzungen zu realisieren.)

Zur Definition der FEISTEL-Funktion $f$ dienen acht sogenannte $S$-Boxen ($S$ = Substitution), die als Wertetabellen einem Eingabewort aus sechs Bit einen vier Bit langen Funktionswert zuordnen; sie beschreiben also Abbildungen von $\mathbb{F}_2^6$ nach $\mathbb{F}_2^4$.

Diese Wertetabellen sind folgendermaßen angeordnet: Das Eingabewort mit seinen sechs Bit wird geschrieben als $(a, m, e)$, wobei $a$ das Anfangsbit, $e$ das Endbit und $m$ die aus vier Bit bestehende Mitte ist. Diese wird als Zahl zwischen null und fünfzehn aufgefaßt, genau wie auch die Ausgabe der $S$-Box. Die $S$-Box wird angegeben durch vier Zeilen, die mit



den verschiedenen Möglichkeiten für das Paar $(a, e)$ indiziert sind, und die für die sechszehn Werte von $m$ die Ausgabewerte enthalten:

**Box 1**

| $a\ e$ | m = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 0 | 14 | 4 | 13 | 1 | 2 | 15 | 11 | 8 | 3 | 10 | 6 | 12 | 5 | 9 | 0 | 7 |
| 0 1 | 0 | 15 | 7 | 4 | 14 | 2 | 13 | 1 | 10 | 6 | 12 | 11 | 9 | 5 | 3 | 8 |
| 1 0 | 4 | 1 | 14 | 8 | 13 | 6 | 2 | 11 | 15 | 12 | 9 | 7 | 3 | 10 | 5 | 0 |
| 1 1 | 15 | 12 | 8 | 2 | 4 | 9 | 1 | 7 | 5 | 11 | 3 | 14 | 10 | 0 | 6 | 13 |

**Box 2**

| $a\ e$ | m = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 0 | 15 | 1 | 8 | 14 | 6 | 11 | 3 | 4 | 9 | 7 | 2 | 13 | 12 | 0 | 5 | 10 |
| 0 1 | 3 | 13 | 4 | 7 | 15 | 2 | 8 | 14 | 12 | 0 | 1 | 10 | 6 | 9 | 11 | 5 |
| 1 0 | 0 | 14 | 7 | 11 | 10 | 4 | 13 | 1 | 5 | 8 | 12 | 6 | 9 | 3 | 2 | 15 |
| 1 1 | 13 | 8 | 10 | 1 | 3 | 15 | 4 | 2 | 11 | 6 | 7 | 12 | 0 | 5 | 14 | 9 |

**Box 3**

| $a\ e$ | m = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 0 | 10 | 0 | 9 | 14 | 6 | 3 | 15 | 5 | 1 | 13 | 12 | 7 | 11 | 4 | 2 | 8 |
| 0 1 | 13 | 7 | 0 | 9 | 3 | 4 | 6 | 10 | 2 | 8 | 5 | 14 | 12 | 11 | 15 | 1 |
| 1 0 | 13 | 6 | 4 | 9 | 8 | 15 | 3 | 0 | 11 | 1 | 2 | 12 | 5 | 10 | 14 | 7 |
| 1 1 | 1 | 10 | 13 | 0 | 6 | 9 | 8 | 7 | 4 | 15 | 14 | 3 | 11 | 5 | 2 | 12 |

**Box 4**

| $a\ e$ | m = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 0 | 7 | 13 | 14 | 3 | 0 | 6 | 9 | 10 | 1 | 2 | 8 | 5 | 11 | 12 | 4 | 15 |
| 0 1 | 13 | 8 | 11 | 5 | 6 | 15 | 0 | 3 | 4 | 7 | 2 | 12 | 1 | 10 | 14 | 9 |
| 1 0 | 10 | 6 | 9 | 0 | 12 | 11 | 7 | 13 | 15 | 1 | 3 | 14 | 5 | 2 | 8 | 4 |
| 1 1 | 3 | 15 | 0 | 6 | 10 | 1 | 13 | 8 | 9 | 4 | 5 | 11 | 12 | 7 | 2 | 14 |

**Box 5**

| $a\,e$ | m = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 0 | 2 | 12 | 4 | 1 | 7 | 10 | 11 | 6 | 8 | 5 | 3 | 15 | 13 | 0 | 14 | 9 |
| 0 1 | 14 | 11 | 2 | 12 | 4 | 7 | 13 | 1 | 5 | 0 | 15 | 10 | 3 | 9 | 8 | 6 |
| 1 0 | 4 | 2 | 1 | 11 | 10 | 13 | 7 | 8 | 15 | 9 | 12 | 5 | 6 | 3 | 0 | 14 |
| 1 1 | 11 | 8 | 12 | 7 | 1 | 14 | 2 | 13 | 6 | 15 | 0 | 9 | 10 | 4 | 5 | 3 |

**Box 6**

| $a\,e$ | m = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 0 | 12 | 1 | 10 | 15 | 9 | 2 | 6 | 8 | 0 | 13 | 3 | 4 | 14 | 7 | 5 | 11 |
| 0 1 | 10 | 15 | 4 | 2 | 7 | 12 | 9 | 5 | 6 | 1 | 13 | 14 | 0 | 11 | 3 | 8 |
| 1 0 | 9 | 14 | 15 | 5 | 2 | 8 | 12 | 3 | 7 | 0 | 4 | 10 | 1 | 13 | 11 | 6 |
| 1 1 | 4 | 3 | 2 | 12 | 9 | 5 | 15 | 10 | 11 | 14 | 1 | 7 | 6 | 0 | 8 | 13 |

**Box 7**

| $a\,e$ | m = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 0 | 4 | 11 | 2 | 14 | 15 | 0 | 8 | 13 | 3 | 12 | 9 | 7 | 5 | 10 | 6 | 1 |
| 0 1 | 13 | 0 | 11 | 7 | 4 | 9 | 1 | 10 | 14 | 3 | 5 | 12 | 2 | 15 | 8 | 6 |
| 1 0 | 1 | 4 | 11 | 13 | 12 | 3 | 7 | 14 | 10 | 15 | 6 | 8 | 0 | 5 | 9 | 2 |
| 1 1 | 6 | 11 | 13 | 8 | 1 | 4 | 10 | 7 | 9 | 5 | 0 | 15 | 14 | 2 | 3 | 12 |

**Box 8**

| $a\,e$ | m = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 0 | 13 | 2 | 8 | 4 | 6 | 15 | 11 | 1 | 10 | 9 | 3 | 14 | 5 | 0 | 12 | 7 |
| 0 1 | 1 | 15 | 13 | 8 | 10 | 3 | 7 | 4 | 12 | 5 | 6 | 11 | 0 | 14 | 9 | 2 |
| 1 0 | 7 | 11 | 4 | 1 | 9 | 12 | 14 | 2 | 0 | 6 | 10 | 13 | 15 | 3 | 5 | 8 |
| 1 1 | 2 | 1 | 14 | 7 | 4 | 10 | 8 | 13 | 15 | 12 | 9 | 0 | 3 | 5 | 6 | 11 |

Diese $S$-Boxen werden zusammengesetzt zu einer Funktion

$$F\colon \mathbb{F}_2^{48} \to \mathbb{F}_2^{32}$$

in der offensichtlichen Weise: Ein Vektor der Länge 48 wird aufgeteilt in acht Vektoren der Länge sechs; auf den ersten davon wird die erste $S$-Box angewandt, auf den zweiten die zweite *usw;* dabei entstehen



acht Vektoren der Länge vier, die zum Ergebnisvektor der Länge 32 zusammengesetzt werden.

Nach diesem Konfusionsschritt folgt noch ein Diffusionsschritt: Die Komponenten des Vektors werden untereinander permutiert mittels einer Permutation aus $S_{32}$, die durch folgende Wertetabelle geben ist:

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 16 | 7 | 20 | 21 | 29 | 12 | 28 | 17 | 1 | 15 | 23 | 26 | 5 | 18 | 31 | 10 |
| 2 | 8 | 24 | 14 | 32 | 27 | 3 | 9 | 19 | 13 | 30 | 6 | 22 | 11 | 4 | 25 |

Die Funktion $F$ wird folgendermaßen eingesetzt: Zunächst wird die rechte Hälfte $R$ der Eingabe der jeweiligen Runde auf 48 Bit vergrößert, indem man einen Vektor $(x_1, \ldots, x_{32})$ ersetzt durch

$$(x_{\tau(1)}, \ldots, x_{\tau(48)})\,,$$

wobei die Werte von $\tau$ der folgenden Tabelle entnommen werden:

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 32 | 1 | 2 | 3 | 4 | 5 | 4 | 5 | 6 | 7 | 8 | 9 | 8 | 9 | 10 | 11 | 12 | 13 |
| 12 | 13 | 14 | 15 | 16 | 17 | 16 | 17 | 18 | 19 | 20 | 21 | 20 | 21 | 22 | 23 | 24 | 25 |
| 24 | 25 | 26 | 27 | 28 | 29 | 28 | 29 | 30 | 31 | 32 | 1 | | | | | | |

Das Ergebnis dieser Aufblähung wird zum Schlüssel der $i$-ten Runde addiert (in $\mathbb{F}_2^{48}$, d.h. die Addition ist ein logisches XOR).

Die kryptographische Funktion dieser Aufblähung ist natürlich wieder eine Diffusion; sie sorgt dafür, daß ein Eingabebit möglichst viele Ausgabebits beeinflußt.

Damit kommen wir zur Verwendung des Schlüssels im Algorithmus. Der Schlüssel hat, wie bereits erwähnt, 56 Bit, wird aber mit 64 Bit gespeichert, wobei jedes achte Bit ein Paritätsbit ist, d.h. die Summe (in $\mathbb{F}_2$) der sieben davorstehenden Bits. Wir numerieren die Schlüsselbits daher von eins bis 64, verwenden aber nur die nicht durch acht teilbaren Indizes.

Aus dem Schlüssel werden zunächst zwei Schlüssel der Länge 28 extrahiert, bestehend aus den folgenden Komponenten:

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 | 58 | 50 | 42 | 34 | 26 | 18 | | |
| 10 | 2 | 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 | 60 | 52 | 44 | 36 | | |

und

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 63 | 55 | 47 | 39 | 31 | 23 | 15 | | 7 | 62 | 54 | 46 | | 38 | 30 | 22 |
| 14 | | 6 | 61 | 53 | 45 | 37 | 29 | 21 | 13 | | 5 | | 28 | 20 | 12 | | 4 |

Die so erhaltenen Teilschlüssel werden vor jeder Runde noch zirkulär nach links verschoben, und zwar vor der $i$-ten Runde um nochmals $a_i$ Bit gegenüber der vorherigen Runde, wobei $a_1$ bis $a_{16}$ die Zahlenfolge

$$1\ 1\ 2\ 2\ 2\ 2\ 2\ 2\ 1\ 2\ 2\ 2\ 2\ 2\ 2\ 1$$

ist. Man beachte, daß die Summe dieser Zahlen gleich 28 ist, jeder der Halbschlüssel wird also in den sechzehn Runden einmal komplett zyklisch verschoben.

Nachdem die beiden Teilschlüssel so präpariert sind und aneinandergehängt einen 56 Bit-Schlüssel $(s_1, \ldots, s_{56})$ bilden, wird daraus ein 48 Bit-Schlüssel für die $i$-te Runde gewählt, bestehend aus den folgenden Komponenten:

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 14 | 17 | 11 | 24 | | 1 | | 5 | | 3 | 28 | 15 | | 6 | 21 | 10 |
| 23 | 19 | 12 | | 4 | 26 | | 8 | 16 | | 7 | 27 | 20 | 13 | | 2 |
| 41 | 52 | 31 | 37 | 47 | 55 | 30 | | 40 | 51 | 45 | 33 | 48 | | | |
| 44 | 49 | 39 | 56 | 34 | 53 | 46 | | 42 | 50 | 36 | 29 | 32 | | | |

Die Rundenschlüssel werden also in einem reinen Diffusionsverfahren aus dem Gesamtschlüssel berechnet.

Die FEISTEL-Funktion $f$ berechnet die Summe aus den 48 aufgeblähten Nachrichtenbits und den 48 Schlüsselbits der $i$-ten Runde und setzt diesen Vektor aus $F_2^{48}$ in $F$ ein; der Funktionswert ist der Wert der FEISTEL-Funktion.

Dieses Spiel wird insgesamt sechzehnmal gespielt, wobei nach der letzten Runde keine Vertauschung von links und rechts mehr stattfindet. Danach wird nur noch die Anfangspermutation rückgängig gemacht; die inverse Permutation hat die Wertetabelle

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 40 | | 8 | 48 | 16 | 56 | 24 | 64 | 32 | 39 | | 7 | 47 | 15 | 55 | 23 | 63 | 31 |
| 38 | | 6 | 46 | 14 | 54 | 22 | 62 | 30 | 37 | | 5 | 45 | 13 | 53 | 21 | 61 | 29 |
| 36 | | 4 | 44 | 12 | 52 | 20 | 60 | 28 | 35 | | 3 | 43 | 11 | 51 | 19 | 59 | 27 |
| 34 | | 2 | 42 | 10 | 50 | 18 | 58 | 26 | 33 | | 1 | 41 | | 9 | 49 | 17 | 57 | 25 |



Man beachte, daß DES (wie jedes (balancierte) FEISTEL-Netzwerk) in jeder Runde nur einen halben Block verändert; die andere Hälfte bleibt erhalten. Schreiben wir den Nachrichtenblock nach Anwendung der Anfangspermutation also in der Form $(m_0, m_1)$, so wird in der $i$-ten Runde $(m_{i-1}, m_i)$ zu $(m_i, m_{i+1})$ mit

$$m_{i+1} = m_{i-1} \oplus f(s_i, m_i)\ ,$$

wobei wieder $s_i$ den Schlüssel der $i$-ten Runde bezeichnet. Das Ergebnis der sechzehn Runden, abgesehen von der Endpermutation, ist dann allerdings nicht $(m_{16}, m_{17})$, sondern $(m_{17}, m_{16})$, da nach der letzten Runde die Hälften nicht mehr miteinander vertauscht werden.

Der Grund dafür liegt in der Entschlüsselung: Wegen

$$m_{i+1} = m_{i-1} \oplus f(s_i, m_i) \Leftrightarrow m_{i-1} = m_{i+1} \oplus f(s_i, m_i)$$

läßt sich die Verschlüsselung bei Kenntnis des Schlüssels leicht rückgängig machen, sogar mit derselben Hardware. Da aber vor dem ersten Verschlüsselungsschritt die Hälften nicht vertauscht werden, sollten sie es dann auch nach dem letzten nicht mehr werden, denn die Entschlüsselung läuft ja rückwärts durch die Runden.

## § 3: Designkriterien und Kryptoanalyse des DES

Die Beschreibung des DES im vorigen Paragraphen läßt den Leser zunächst wohl mit ziemlicher Verwirrung zurück, und es erscheint schwierig, irgendeine Aussage über die kryptographische Sicherheit des Verfahrens zu machen. In der Tat wurde diese von Anfang an sehr kontrovers diskutiert.

### a) Geschichtliche Entwicklung

Als das damalige *National Bureau of Standards* der USA (heute National Institute of Standards and Technology, NIST) im Januar 1977 den DES als Standard veröffentlichte (mit einer auf zehn Jahre veranschlagten Laufzeit) enthielt das Dokument im wesentlichen nur die hier reproduzierten Angaben; sowohl IBM als auch die National Security

Agency (NSA) lehnten es ab, die Kriterien zu benennen, nach denen die $S$-Boxen und die Permutationen konzipiert worden waren.

Dies führte schnell auf den Verdacht, daß DES möglicherweise eine nur IBM und NSA bekannte „Falltür“ enthält, mit deren Hilfe eine Entschlüsselung ohne Schlüssel mit vertretbarem Aufwand durchgeführt werden kann. Außerdem gab es bereits damals Kritik an der mit 56 Bit sehr kurzen Schlüssellänge: Das Vorgängersystem LUCIFER hatte eine Schlüssellänge von 128 Bit, im militärischen Bereich waren Systeme mit mehr als zehn mal so langen Schlüsseln nichts Ungewöhnliches.

M.E. HELLMAN: A cryptanalytic time-memory tradeoff, *IEEE Trans. Inf. Theory* **26** (1980), 401–406

schlug ein Verfahren vor, mit dem durch eine Kombination von Vorberechnungen und Probieren die Komplexität der Schlüsselsuche von $2^{56}$ mit großer Erfolgswahrscheinlichkeit auf ungefähr die Kubikwurzel dieser Zahl reduziert werden konnte; als Baupreis seiner Maschine schätzte er zehn Millionen Dollar, als Zeitrahmen für die Vorberechnungen ungefähr ein Jahr. Da die NSA erheblich größere Geldmittel als nur zehn Millionen Dollar einsetzen kann, bestärkte dies den Verdacht, daß sie DES selbst dann knacken kann, wenn der Algorithmus keine Falltür enthalten sollte.

Im Laufe der Jahre wurden einige der Designkriterien durch *reverse engineering* gefunden; einige dann auch freiwillig veröffentlicht. Erst 1994 veröffentlichte einer der ursprünglichen Entwickler bei IBM die, wie er sagt, vollständige Liste der kryptographisch relevanten Kriterien in

D. COPPERSMITH: The Data Encryption Standard (DES) and its strength against attacks, *IBM J. Res. Develop.* **38** (1994), S. 243–250

– nachdem die kryptoanalytische Technik, gegen die diese Kriterien schützen sollten, auch in der offenen Literatur erschienen war. Dabei zeigte sich, daß DES mit seinen nur 56 Bit zumindest gegen diese Technik eine eher größere Sicherheit bietet als Lucifer mit seinen 128 Bit und daß die Sicherheit von DES nicht unbedingt erhöht würde, indem man für jede der sechzehn Runden einen neuen 48 Bit-Schlüssel verwendet,



so daß man insgesamt eine Schlüssellänge von $16 \times 48 = 762$ Bit hätte. NSA hielt diese Technik damals für so wichtig für den Angriff auf gegnerische Systeme, daß die speziell dagegen eingesetzten Designkriterien „aus Gründen der nationalen Sicherheit“ geheimgehalten wurden.

Die Technik, um die es hier geht, war bei IBM um 1974 unter dem Namen *T attack* bekannt; in der offenen Literatur erschienen erste Ansätze dazu ab etwa 1988, vollständige Beschreibungen erschienen ab 1990 unter dem Namen *differentielle Kryptoanalyse*. Bevor wir sie genauer betrachten, wollen wir uns zunächst die inzwischen bekannten Designkriterien des DES ansehen.

## b) Designkriterien

D. COPPERSMITH nennt in der oben zitierten Arbeit folgende Designkriterien für die $S$-Boxen (und sagt, daß dies *alle* kryptographisch relevanten gewesen seien; der Rest habe nur mit Implementierungsfragen zusammengehängt):

*(S1)* Jede $S$-Box hat sechs Eingabe- und vier Ausgabebits.

*(S2)* Kein Ausgabebit einer $S$-Box sollte zu nahe bei einer linearen Funktion der Eingabebits liegen.

*(S3)* Bei festgehaltenem linken und rechtem Bit der Eingabe sollte jeder der sechzehn möglichen Ausgabewerte genau einmal vorkommen.

*(S4)* Wenn sich zwei Eingaben einer $S$-Box um genau ein Bit unterscheiden, müssen sich die Ausgaben um mindestens zwei Bit unterscheiden.

*(S5)* Wenn sich zwei Eingaben einer $S$-Box genau in den beiden mittleren Bits unterscheiden, müssen sich die Ausgaben um mindestens zwei Bit unterscheiden.

*(S6)* Wenn sich zwei Eingaben einer $S$-Box in ihren beiden Anfangsbits, nicht aber in ihren beiden Endbits unterscheiden, müssen die Ausgaben verschieden sein.

*(S7)* Für jede von null verschiedene Differenz $\Delta$ zwischen zwei Eingaben dürfen höchstens acht der 32 Paare mit Differenz $\Delta$ auf dieselbe Differenz zwischen den Ausgaben führen.

*(S8)* Ähnlich zu *(S7)*, aber mit stärkeren Eigenschaften für den Fall gleicher Ausgaben, wenn in einer Runde drei $S$-Boxen „aktiv" sind. *(s.u.)*

Für die Permutation aus $S_{32}$, die in jeder FEISTEL-Funktion als Abschluß ausgeführt wird, sollten folgende Bedingungen erfüllt sein:

*(P1)* Die vier Ausgabebits einer $S$-Box werden so verteilt, daß in der nächsten Runde zwei von ihnen mittlere Bits der Eingabe einer $S$-Box sind und die beiden anderen nicht (d.h. die kommen an Position 1, 2, 5 oder 6).

*(P2)* Die vier Ausgabebits einer $S$-Box sind in der nächsten Runde Eingaben zu sechs verschiedenen $S$-Boxen; keine zwei von ihnen sind Eingabe derselben $S$-Box.

*(P3)* Für zwei (nicht notwendigerweise verschiedene) $S$-Boxen $j, k$ gilt: Wenn ein Ausgabebit von $j$ als eines der beiden mittleren Bits an $k$ weitergegeben wird, kann kein Ausgabebit von $k$ als mittleres Bit an $j$ weitergegeben werden. Insbesondere darf also kein Ausgabebit von $j$ an $j$ selbst als mittleres Bit weitergegeben werden.

Der Sinn einiger dieser Kriterien ist unmittelbar einsichtig: *(S1)* etwa kommt daher, daß mit der Technologie von 1974 größere $S$-Boxen dazu geführt hätten, daß man den Algorithmus nicht auf einem Chip untergebracht hätte.

*(S2)* ist selbstverständlich: Da die $S$-Boxen der einzige nichtlineare Bestandteil des Algorithmus sind, müssen sie nichtlinear sein; ansonsten hätten wir eine (leicht zu knackende) HILL-Chiffre. Wenn einzelne Bits lineare Funktionen der Eingabebits wären, hätten wir möglicherweise für einzelne Ausgabebits des Algorithmus lineare Zusammenhänge mit den Eingabebits, was dazu führen würde, daß man zumindest einen Teil der Chiffre als HILL-Chiffre betrachten kann und damit die Komplexität des Algorithmus reduziert. Ähnlich verhält es sich, wenn Funktionen



nicht exakt, aber doch ungefähr linear sind – mehr dazu gleich bei der *linearen Kryptanalyse*.

Die restlichen Kriterien dienen in erster Linie zur Förderung der Diffusion: Für zwei verschiedene Eingaben $W, W'$ in Runde $i$ sagen wir, eine $S$-Box sei *aktiv*, wenn sie für $W$ und $W'$ verschiedene Ausgaben liefert. Es muß nicht in jeder Runde aktive $S$-Boxen geben, aber die obigen Kriterien sollen dafür sorgen, daß im Durchschnitt über alle Runden möglichst viele $S$-Boxen pro Runde aktiv sind; wie man zeigen kann, sind es im Durchschnitt mindestens 1,6.

(Bei *(S7)* sind die Zahlen, so wie sie genannt wurden, offensichtlich um den Faktor zwei zu klein: Es gibt 64 Paare mit vorgegebener Differenz $\Delta$, und für $\Delta \neq 0$ dürfen dann wohl höchstens 16 davon auf denselben Ausgabewert führen.)

Bevor wir solche Fragen vertiefen können, müssen wir uns zunächst mit der kryptoanalytischen Attacke beschäftigen, vor der dies schützen soll:

### c) Differentielle Kryptoanalyse

Ihre Grundidee besteht darin, daß man nicht von einzelnen Klartextblöcken ausgeht, sondern von Paaren $(W, W')$ aus zwei Klartextblöcken. Diese werden aufgefaßt als Elemente von $\mathbb{F}_2^{64}$; da über dem Körper mit zwei Elementen Addition gleich Subtraktion ist, bezeichnen wir die Differenz zwischen den beiden Nachrichten als $W \oplus W'$. Praktisch handelt es sich hier einfach um das bitweise XOR zwischen den beiden Blöcken.

DES unterzieht die beiden Worte zunächst der Anfangspermutation; da XOR eine bitweise Operation ist, wird dabei auch die Differenz $W \oplus W'$ dieser Permutation unterzogen. Danach werden die rechten Hälften der entstandenen Nachrichten betrachtet; ihre Differenz ist natürlich einfach die rechte Hälfte der permutierten Differenz. Die entstandenen 32 Bit-Worte werden durch Bitauswahl auf 48 Bit Worte $V, V'$ aufgebläht; auch diese Aufblähung ist kompatibel mit der Differenzbildung.

Als nächstes kommt der Schlüssel ins Spiel; sowohl $V$ als auch $V'$ werden zum 48-Bit Schlüssel $s_1$ der ersten Runde addiert; dann gehen

die Ergebnisse $V \oplus s_1$ und $V' \oplus s_1$ in acht 6 Bit Stücke aufgespalten in die acht $S$-Boxen. Die Differenz zwischen den beiden Eingaben ist

$$(V \oplus s_1) \oplus (V' \oplus s_1) = (V \oplus V') \oplus (s_1 \oplus s_1) = V \oplus V' ,$$

d.h. der Schlüssel ist herausgefallen.

Nun kommen die $S$-Boxen ins Spiel. Falls diese linear wären, wäre die Differenz ihre Ausgabe für zwei gegebene Eingabewerte nur von der Differenz der Eingabewerte abhängig, aber da es gerade der Zweck der $S$-Boxen ist, die Verschlüsselung nichtlinear zu machen, können wir natürlich nicht erwarten, daß wir hier auch nur bei einer einzigen $S$-Box eine lineare Funktion finden: Schon die ersten experimentellen Untersuchungen von DES befaßten sich mit etwaigen linearen Zusammenhängen zwischen einzelnen Ausgabebits sowohl einer $S$-Box wie auch des gesamten DES und der jeweiligen Eingabe, und keine konnte eine lineare Funktion finden.

Nach der Anwendung der $S$-Boxen können wir also nicht mehr sagen, was die Differenz der Ausgabewerte ist, obwohl wir die Differenz der Eingabewerte auch unabhängig vom Schlüssel kennen.

Trotzdem zeigt sich, daß wir zumindest gewisse Informationen über die Differenz haben: Bei sechs Eingabebits und vier Ausgabebits pro $S$-Box müssen von den $2^6 = 64$ Eingabepaaren $(E, E')$ mit einer gegebenen Differenz $\Delta E$ im Durchschnitt jeweils vier auf jede der sechzehn möglichen Differenzen $\Delta A$ der Ausgabewerte $A$, $A'$ führen. Im Einzelnen gibt es allerdings beträchtliche Schwankungen:

Für $\Delta E = 0$ ist natürlich auch $\Delta A = 0$, denn dasselbe Wort kann nicht auf zwei verschiedene Weisen verschlüsselt werden. Aber auch für andere Werte von $\Delta E$ gibt es keine Gleichverteilung der Ausgabewerte: Für $\Delta E = 100100$ etwa ergibt sich für die (dezimal geschriebenen) Differenzen $\Delta A$ folgende Verteilung:

| $\Delta A$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Fälle* | 12 | 0 | 0 | 2 | 2 | 2 | 2 | 0 | 14 | 14 | 2 | 0 | 2 | 6 | 2 | 4 |

Wir haben also hier, wie auch bei den anderen Differenzen und anderen $S$-Boxen eine ziemlich inhomogene Verteilung. (Die Fallanzahlen für jede $S$-Box und jede Ausgabedifferenz sind aufgelistet im Anhang von



E. BIHAM, A. SHAMIR: Differential Cryptanalysis of the Data Encryption Standard, *Springer,* 1993;

in diesem Buch ist die differentielle Kryptoanalyse des DES einschließlich der Verallgemeinerung auf andere Systeme vollständig beschrieben.)

Auf die Ausgabe der acht $S$-Boxen wird noch eine Permutation angewandt; die ist wieder mit Differenzbildung kompatibel. Falls wir also die Differenzen der Ausgaben der $S$-Boxen kennen, bereitet diese Permutation keine Schwierigkeiten und wir kennen die Eingabedifferenzen für die zweite Runde, mit der wir genauso weiter verfahren können.

Tatsächlich kennen wir die Ausgaben der acht $S$-Boxen der ersten Runde natürlich nicht; wir können nur für jede einzelne $S$-Box eine Wahrscheinlichkeitsverteilung der Ausgabedifferenzen angeben. Für einzelne Bits oder Bitgruppen kommen wir dabei durchaus auf recht ansehnliche Wahrscheinlichkeiten: Im obigen Beispiel für die Eingabedifferenz 100100 etwa ist mit jeweils einer Wahrscheinlichkeit von 14 : 64, also in mehr als 20% aller Fälle, die Ausgabedifferenz gleich acht oder neun, binär geschrieben also 0100 oder 0101. Die Wahrscheinlichkeit dafür, daß die ersten drei Bits der Ausgabedifferenz gleich 010 sind, ist also 28 : 64 = 7 : 16, und wenn wir uns auf das erste und dritte Bit beschränken, kommen wir auf eine Wahrscheinlichkeit von 40 : 64 = 5 : 8 dafür, daß die Ausgabedifferenz die Form $0x0y$ hat. Das dritte Bit schließlich ist in 52 der 64 möglichen Fällen gleich null, so daß wir zumindest dieses eine Bit mit der recht hohen Wahrscheinlichkeit von 13 : 16 kennen.

Der Sinn eines FEISTEL-Netzwerks ist es, uns solche Informationen, die wir für eine Runde haben, in den weiteren Runden durch Konfusion und Diffusion weitestgehend zu zerstören. Man wird erwarten, daß dies nicht für alle Klartextdifferenzen gleich gut gelingt; die Idee hinter der differentiellen Kryptoanalyse ist, sich auf die zu konzentrieren, bei denen es möglichst schlecht gelingt. Wir müssen uns also genauer anschauen, wie eine Klartextdiffernz durch die Runden geht.

**Definition:** Eine *$r$-Rundencharakteristik* ist eine Folge

$$\Delta = (\delta_0, \delta_1, \ldots, \delta_r)$$

von Elementen aus $\mathbb{F}_2^{64}$.
Ein Klartextpaar $(x_0, y_0) \in \mathbb{F}_2^{64} \times \mathbb{F}_2^{64}$ *gehört zur Charakteristik* $\Delta$, wenn für die Paare $(x_i, y_i)$ der Ausgaben der $i$-ten Runde gilt: $x_i \oplus y_i = \delta_i$ für $i = 0, \ldots, r$.
Die *Wahrscheinlichkeit* einer Charakteristik ist die Wahrscheinlichkeit dafür, daß ein Klartextpaar $(x_0, y_0)$ mit Differenz $\delta_0$ zur Charakteristik $\Delta$ gehört.

Natürlich sind die Wahrscheinlichkeiten der meisten Charakteristiken sehr gering: In einem idealen Kryptosystem wären alle Ausgabewerte einer Runde gleich wahrscheinlich, die Wahrscheinlichkeit einer $r$-Rundencharakteristik sollte also im Mittel bei $1/2^{-64\times r}$ liegen, und schon die Wahrscheinlichkeit dafür, daß überhaupt ein Klartextpaar mit gegebener Differenz nach $r$-Runden eine vorgegebene andere Differenz hat, sollte im allgeinen bei nur etwa $2^{-64}$ liegen.

Wenn wir eine gute $r$-Rundencharakteristik gefunden haben, deren Wahrscheinlichkeit deutlich besser ist als $2^{-64}$, können wir daher ziemlich sicher sein, daß ein zufällig gewähltes Klartextpaar $(x_0, y_0)$ mit Anfangsdifferenz $\delta_0$ und Enddifferenz $\delta_r$ in den $r$ Runden so verschlüsselt wurde, wie es der Charakteristik entspricht. Ist etwa $p = 2^{-44}$, sollte die Wahrscheinlichkeit für alles andere bei nur etwa $2^{-20}$ oder etwa eins zu einer Million liegen. Konkret heißt das: Wir kennen die Differenzen $x_i \oplus y_i$ mit hoher Wahrscheinlichkeit und können die Verschlüsselung durch die $r$ Runden verfolgen.

Zunächst brauchen wir aber Charakteristiken. Für nur eine Runde ist das sehr einfach: Für jeden Halbblock $d_0 \in \mathbb{F}_2^{32}$ führt der mit 32 Nullen auf 64 Bit aufgefüllte Block $\delta_0$ auf die Charakteristik $(\delta_0, \delta_0)$ mit Wahrscheinlichkeit eins: Sind nämlich $(m_0, m_1)$ und $(m_0', m_1)$ zwei Klartexte mit (nach der hier stets ignorierten Anfangspermutation) gleicher rechter Hälfte, so wird die linke Hälfte ersetzt durch

$$m_2 = m_0 \oplus f(s_1, m_1) \quad \textit{bzw.} \quad m_2' = m_0' \oplus f(s_1, m_1)\,,$$



und die Differenz ist

$$m_2 \oplus m_2' = \big(m_0 \oplus f(s_1, m_1)\big) \oplus \big(m_0' \oplus f(s_1, m_1)\big) = m_0 \oplus m_0' = d_0\,,$$

wie wir das schon oben gesehen haben. Nach Durchgang durch die erste Runde haben wir also die beiden Paare $(m_2, m_1)$ und $(m_2', m_1)$, deren Differenz wieder $\delta_0$ ist.

Leider können wir diese Charakteristik nicht iterieren, denn nach der ersten Runde werden ja die beiden Hälften vertauscht, so daß wir dann $(m_1, m_2')$ und $(m_1, m_2')$ haben, worüber wir nicht soviel sagen können, da nun die FEISTEL-Funktionen verschiedene Werte liefern.

Um trotzdem zu einer Zweirundencharakteristik zu kommen, benutzen wir die Nichtinjektivität der FEISTEL-Funktion $f$: Wir suchen zwei Halbblöcke $m_0$ und $m_0'$ derart, daß $f(s, m_0) = f(s, m_0')$ für möglichst viele Schlüssel $s$; die Differenz $m_0 \oplus m_0'$ bezeichnen wir mit $d_0$.

Zwei beliebige Klartexte der Form $(m_0, m_1)$ und $(m_0', m_1)$ mit Differenz $m_0 \oplus m_0' = d_0$ gehen in der ersten Runde nach $(m_2, m_1)$ und $(m_2', m_1)$ mit

$$m_2 = m_0 \oplus f(s_1, m_1) \quad \text{und} \quad m_2' = m_0' \oplus f(s_1, m_1)\,,$$

danach werden die linke und die rechte Hälfte vertauscht, so daß die Eingaben zur zweiten Runde gleich $(m_1, m_2)$ und $(m_1, m_2')$ sind, wobei $m_2$ und $m_2'$ die Differenz $d_0$ haben. In der zweiten Runde gehen die beiden Klartexte dann nach

$$\big(m_1 \oplus f(s_2, m_2), m_2\big) \quad \text{und} \quad \big(m_1 \oplus f(s_2, m_2'), m_2'\big)\,.$$

Da $m_2$ und $m_2'$ Differenz $d_0$ haben, ist dabei mit einer gewissen Wahrscheinlichkeit

$$f(s_2, m_2) = f(s_2, m_2')\,.$$

Falls dem so sein sollte, haben wir $(0, d_0)$ als Differenz zwischen den Ausgabewerten. Es gibt daher eine Zweirundencharakteristik der Form

$$\big((d_0, 0), (d_0, 0), (0, d_0)\big)\,,$$

die mit einer gewissen Wahrscheinlichkeit $p$ auftritt. Diese Charakteristik kann offensichtlich beliebig oft iteriert werden, denn bevor ihre Ausgabewerte in die nächste Runde gehen, werden die linke und die

rechte Hälfte vertauscht, so daß wir wieder die Ausgangsdifferenz haben. Damit haben wir für jedes $n$ eine $n$-Rundencharakteristik gefunden; ihre Wahrscheinlichkeit ist $p^{\lfloor n/2 \rfloor}$, wobei die GAUSS-Klammer $[x]$ die größte ganze Zahl $\leq x$ bezeichnet.

Diese Charakteristik ist umso nützlicher, je größer $p$ ist. Wie sich zeigt, kann $p$ nur dann größer als null sein, wenn mindestens drei benachbarte $S$-Boxen aktiv sind; die größten Wahrscheinlichkeiten sind also zu erwarten bei *genau* drei aktiven $S$-Boxen. Systematisches Probieren zeigt, daß der beste erreichbare Wert dann

$$p = \frac{14}{64} \cdot \frac{8}{64} \cdot \frac{10}{64} = \frac{35}{8192} \approx 0{,}004272461 \approx \frac{1}{234} \approx 2^{-7{,}870716983}$$

ist. Er wird erreicht für

$$d_0 = (19\,60\,00\,00)_{\text{hex}} \quad \text{und} \quad d_0 = (1B\,60\,00\,00)_{\text{hex}}\,.$$

Damit haben wir also für beliebiges $n$ eine $n$-Rundencharakteristik gefunden; leider ist sie aber nicht für jedes $n$ brauchbar: Für 16 Runden ist ihre Wahrscheinlichkeit nur etwa

$$2^{-7{,}870716983 \times 8} \approx 2^{-62{,}96573586}\,,$$

wir bräuchten also mindestens $2^{63}$ Klartextpaare bekannter Differenz, um eines zu dieser Charakteristik zu finden, während wir mit nur $2^{56}$ Versuchen alle Schlüssel durchprobieren könnten. Tatsächlich reichten sogar bereits $2^{55}$ Versuche, denn nimmt man das Einserkomplement von Schlüssel und Klartext, so entsteht das Einserkomplement des Schlüsseltexts. Deshalb sind auch die 14- und die 15-Rundencharakteristik, deren Wahrscheinlichkeiten bei

$$2^{-7{,}870716983 \times 7} \approx 2^{-55{,}09501888}$$

liegen, nicht sonderlich interessant; erst die 13-Rundencharakteristik liefert mit

$$p \approx 2^{-7{,}870716983 \times 6} \approx 2^{-47{,}22430190}$$

eine halbwegs interessante Wahrscheinlichkeit.

Wählen wir ein Halbwort $m_0$ derart, daß $m_0 \oplus f(s, m_1)$ für möglichst viele Schlüssel $s$ gleich ist! Für einen solchen Schlüssel sind dann

$$m_2 = m_0 \oplus f(s, m_1) \quad \text{und} \quad m_2' = m_0' \oplus f(s, m_1)$$



Außerdem interessieren wir uns nicht für die Wahrscheinlichkeitsverteilung der Chiffretexte – Chiffretext ist schließlich das, was wir immer haben – sondern für Klartext oder besser noch den Schlüssel bei gegebenem Chiffretext.

Dazu nutzen wir aus, daß Eingabewert der $S$-Boxen nicht der ersten Runde nicht der Klartext ist, sondern der mit gewissen Schlüsselbits geXORte Klartext. Wenn wir nun für ein Klartextpaar mit gegebener Differenz die Ausgabedifferenz kennen, haben wir die möglichen Eingabepaare der $S$-Boxen, deren Differenzen ja genau dieselben sind wie für das Klartextpaar, von 64 auf eine erheblich kleinere Zahl reduziert. Für jedes dieses möglichen Paare können wir die entsprechenden Schlüsselbits durch XOR mit dem tatsächlichen Klartext berechnen und haben somit eine relativ kleine Anzahl potentieller Schlüsselteile. Wenn wir das ganze für hinreichend viele Klartextpaare wiederholen, sollte der für die jeweilige $S$-Box zuständige Schlüsselanteil relativ bald eindeutig feststehen.

DES mit nur einer Runde ist auf diese Weise also relativ einfach zu entschlüsseln, falls wir genügend viele Paare von Klartext mit fester Differenz haben. Diese können wir uns nur verschaffen durch eine Attacke mit wählbarem Klartext, also der schwierigsten Form der Attacke.

Auch diese Attacke liefert aber nicht die Ausgabedifferenzen der ersten Runde, sondern nur die der *letzten*. Der Ansatz der differentiellen Kryptoanalyse des DES ist daher folgender:

1. Man wähle eine geeignete Differenz zwischen Klartexten.
2. Dazu erzeuge man hinreichend viele Paare von Klartextblöcken mit dieser Differnz, verschlüssele sie und behalte nur die so berechneten Chiffretextpaare.
3. Durch Analyse der Klartextdifferenzen und des Verhaltens der $S$-Boxen in den verschiedenen Runden bestimme man die zu erwartende Wahrscheinlichkeitsverteilung der Eingabedifferenzen der letzten Runde.

Differentielle Kryptanalyse war der erste Ansatz, DES mit geringerem Aufwand als der vollständigen Durchsuchung des Schlüsselraums zu zu brechen. Da aber, wie bereits erwähnt, die Designer des DES die

differentielle Kryptanalyse schon kannten lange bevor sie in der offenen Literatur auftauchte und den Algorithmus so gut wie möglich dagegen immun machten, ist diese Attacke nicht sehr praktikabel: Selbst bei Angriffen mit frei wählbarem Klartext braucht man über $2^{40}$ Paare aus Klartext und Chiffretext, um den Schlüssel zu finden.

### d) Lineare Kryptanalyse

Eine leichte Verbesserung bietet die kurz später entdeckte *lineare Kryptanalyse:* Zwar sind die Ausgabebits der $S$-Boxen nach Design-Kriterium ($S2$) auch nicht näherungsweise lineare Funktionen der Eingabebits, aber es kann dennoch vorkommen, daß man eine Linearkombination von Ausgabebits mit einer Wahrscheinlichkeit, die deutlich über 50% liegt durch eine Linearkombination der Eingabebits vorhersagen kann. Mit hinreichend vielen Paaren aus Klartext und Chiffretext Läßt sich dadurch ein niedrigdimensionaler affiner Unterraum des Schlüsselraums $\mathbb{F}_2^{56}$ finden, in dem der Schlüssel mit hoher Wahrscheinlichkeit liegen muß. Die vollständige Durchsuchung dieses Unterraums ist unproblematisch, so daß der Schlüssel mit hoher Wahrscheinlichkeit gefunden werden kann.

Auch für diese Attacke sind allerdings unrealistisch viele Paare aus Klartext und Chiffretext erforderlich (in einer Variante genügen noch mehr reine Chiffretexte), so daß der Gesamtaufwand nicht wirklich geringer sein dürfte als die vollständige Durchsuchung des Schlüsselraums.

Nach allem, was in der offenen Literatur bekannt ist, gibt es also zum Knacken des DES keine wesentlich bessere Alternative zur vollständigen Durchsuchung des Schlüsselraums.

### e) DES-Cracker

Der erste in der offenen Literatur dokumentierte realistische Angriff auf DES war denn auch die vollständige Durchsuchung des Schlüsselraums. Eine amerikanische Bürgerrechtsorganisation, die *Electronic Frontier Foundation (EFF)*, konstruierte eine Maschine mit Spezialhardware zum Knacken von DES mit Chiffretext allein.



Die *Electronic Frontier Foundation* wurde 1990 nach dem großen *Hacker Crackdown* in den USA gegründet; Initiatoren waren unter anderem JOHN PERRY BARLOW, bekannt vor allem durch die Lieder, die er für *The Grateful Dead* schrieb, JOHN GILMORE, einer der Pioniere sowohl von *Sun Microsystems* als auch der *Free Software Foundation*, MITCHELL KAPOR, der Gründer von *Lotus*, sowie STEVE WOZNIAK, einer der beiden Gründer von *Apple*.

Ihr Ansatz ist im wesentlichen der unseres guten alten Feinds, des BAYESschen Gegners: Kodiert man einen englisch- oder deutschsprachigen Klartext im ASCII-Code sollten dem BAYESschen Gegner ein bis zwei Blöcke Chiffretext ausreichen, um den Schlüssel zu finden.

Natürlich verfügen selbst die vier obengenannten Gründer der *Electronic Frontier Foundation* nicht über die unbegrenzten Mittel, die der BAYESsche Gegner einsetzen kann; verglichen mit vielen anderen Gegnern verfügt aber doch jeder von ihnen über beträchtliche Mittel. Trotzdem war das 1997 begonnene und 1998 beendete DES-Cracker-Projekt kein Angriff ohne Rücksicht auf die Kosten: Die *Electronic Frontier Foundation* wollte gerade zeigen, daß DES auch mit begrenzten Mitteln geknackt werden kann. Aus diesem Grund wurde der Ansatz des BAYESschen Gegners an mehreren Stellen optimiert:

Zunächst ist es nicht notwendig, wirklich für *jeden* Schlüssel die bedingte Wahrscheinlichkeit auf Grund des Chiffretexts zu bestimmen: In vielen Fällen werden bei der Entschlüsselung nicht druckbare Zeichen entstehen, so daß schon nach wenigen Byte klar ist, daß die Wahrscheinlichkeit des Schlüssels null ist.

DES Cracker beginnt daher mit der Aussonderung unmöglicher Schlüssel durch massiv parallele Hardware: Die Maschine arbeitet mit zwei Blöcken Chiffretext; sie hat als Kern von EFF entwickelte ASICs *application specific integrated circuits)*, die einen 64-Bit-Block mit einem vorgegebenen Schlüssel dechiffrieren können und die Bytes des Ergebnisses auf vom Benutzer einstellbare Bitmuster überprüfen – beispielsweise darauf, ob es sich um ASCII-Codes druckbarer Zeichen handelt. Nur wenn alle Bytes den gewählten Kriterien genügen, wird auch der zweite Block entsprechend untersucht, und wenn auch hier kein im

Klartext unmögliches Byte auftaucht, wird der Schlüssel zur weiteren Untersuchung an einen die Maschine steuerenden PC weitergegeben, der eine genauere Untersuchung gemäß dem Ansatz des BAYESschen Gegners durchführt.

Von den 256 möglichen ASCI-Werten sind etwa ein Viertel druckbare Zeichen; da DES eine Ausgabe liefert, die sich nur wenig von einer Zufallsfolge unterscheidet, wird ein Block nur mit einer Wahrscheinlichkeit von etwa $1 : 4^8 = 1 : 65536$ den Test bestehen; für zwei Blöcke liegt die Wahrscheinlichkeit entsprechend bei $1 : 4^{16} = 1 : 2^{32}$. Von den $2^{56}$ zu untersuchenden Blöcken werden also nur etwa

$$2^{56-32} = 2^{24} = 16\,777\,216$$

an den PC weitergegeben, und die Untersuchung von etwa 17 Millionen Klartextkandidaten ist kein Problem für einen Standard-PC.

Der hauptsächliche Rechenaufwand liegt in der Voruntersuchung der Blöcke durch die ASICs; je nachdem, wie viele von diesen parallel arbeiten, kann dies mehr oder weniger schnell gehen.

Die tatsächlich gebaute Maschine enthält $1536 = 3 \times 2^9$ ASICs, von denen jedes aus 24 parallel arbeitenden Sucheinheiten besteht; insgesamt können also jeweils 36 864 Schlüssel parallel untersucht werden. Jede Sucheinheit kann zweieinhalb Millionen Schlüssel pro Sekunde untersuchen, die gesamte Maschine also etwas über 92 Milliarden.

Insgesamt müssen $2^{56}$ Schlüssel untersucht werden; im Mittel wird man nach $2^{55}$ Versuchen den richtigen gefunden haben. Dafür braucht man

$$\frac{2^{55}}{92\,160\,000\,000} \approx 390\,937 \text{ Sekunden} \approx 108{,}5 \text{ Stunden} \approx 4{,}5 \text{ Tage}\,.$$

Die Maschine ist skalierbar: Jeweils 64 Chips sitzen auf einem Board und 12 Boards in einer Chassis (einer ehemaligen SUN); die gebaute Maschine besteht aus dem steuernden PC zusammen mit zwei Chassis; der PC könnte aber auch mit deutlich mehr als zwei Chassis arbeiten und auch mehrere PCs wären denkbar.

In der gebauten Version kostete DES Cracker 210 000 $, wovon 80 000 $ Entwicklungskosten waren; der Bau eines zweiten Exemplars wäre also



für 130 000 $ möglich, wobei der Preis bei Serienproduktion wohl noch deutlich gesenkt werden könnte. Mit einer Investition in Größenordnung einer Million Dollar käme man also wohl in den Bereich weniger Stunden für das Auffinden eines Schlüssels.

Da eine Million Dollar auch für Geheimdienste kleiner Länder und (etwas kreative Buchhaltung vorausgesetzt) Großunternehmen kein Problem sind, sollte damit klar sein, daß DES auf keinen Fall mehr heutigen Sicherheitsanforderungen genügt.

Die EFF veröffentlichte sowohl die komplette Hardware-Spezifikation als auch die Software von DES-Cracker; in gedruckter Form findet man sie im Buch

ELECTRONIC FRONTIER FOUNDATION: *Cracking DES. Secrects of Encryption Research, Wiretap Politics & Chip Design*, O'Reilly, 1998

Online ist das Buch unter anderem verfügbar unter

`http://cryptome.org/cracking-des.htm` ;

die DES Cracker Seite der EFF ist

`www.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/` .

## § 4: Modifikationen

### a) Mehrfacher DES

Viele Unternehmen, insbesondere im Bankenbereich, haben viel Geld in DES-Hardware investiert und haben daher wenig Interesse, auf ein neues Verfahren umzusteigen – ganz abgesehen davon, daß 1998 noch kein allgemein anerkannter Nachfolgealgorithmus zur Verfügung stand. (Mit der heutigen Situation werden wir uns im nächsten Kapitel beschäftigen.)

Deshalb bot sich ein Verfahren an, das bereits kurz nach Normierung des DES vorgeschlagen wurde, um den kleinen Schlüsselraum zu vergrößern: Man verschlüssele mehrfach mit verschiedenen Schlüsseln.

Eine zweifache Verschlüsselung hängt von $2^{112}$ statt von $2^{56}$ Schlüsselbit ab; der Aufwand für eine Durchsuchung des gesamten Schlüsselraums steigt also um den Faktor $2^{56}$ gegenüber dem gewöhnlichen DES, was eine Maschine nach Art des DES Crackers nach heutigen Stand der Technik nicht mehr in akzeptabler Zeit durchführen kann.

Eine mehrfache Verschlüsselung bietet allerdings nur dann Vorteile, wenn die Hintereinanderausführung zweier DES-Verschlüsselungen nicht äquivalent zur einfachen DES-Verschlüsselung mit einem anderen Schlüssel ist.

Um dies zu untersuchen, müssen wir noch einmal zurück zur grundsätzlichen Struktur von Blockchiffren: Blockchiffren sind Permutationen auf der Menge aller Blöcke; im Falle von DES also auf der Menge aller bijektiver Abbildungen von der Menge aller 64-Bit-Blöcke auf sich selbst.

Die $2^{56}$ durch DES definierten Permutationen bilden natürlich nur eine winzige Teilmenge der Gruppe aller $2^{64}!$ solcher Permutationen; falls diese Teilmenge eine Gruppe sein sollte (oder in einer relativ kleinen Untergruppe liegt), kann die mehrfache Anwendung von DES keine (oder nur wenig) zusätzliche Sicherheit bieten.

Wir brauchen daher Informationen darüber, wie groß die kleinste Gruppe ist, die alle $2^{56}$ DES-Permutationen enthält. Über deren genaue Struktur und Elementanzahl ist leider nichts bekannt, man kann aber immerhin untere Schranken angeben: Für jeden einzelnen DES-Schlüssel kann man die zugehörige Substitution so lange wiederholen, bis die Identität entsteht; da $2^{64}!$ eine zwar große, aber endliche Zahl ist, muß dies nach endlich vielen Schritten der Fall sein.

Angenommen, bei solchen Berechnungen mit verschiedenen Schlüsseln ergaben sich die Ordnungen $n_1, n_2, \ldots, n_r$. Dann enthält die kleinste Gruppe, in der alle DES-Substitutionen liegen, zyklische Untergruppen der Ordnungen $n_1, \ldots, n_r$. Nach einem einfachen Satz der Gruppentheorie, dem Satz von LAGRANGE, müssen die Zahlen $n_1, \ldots, n_r$ dann die Ordnung der gesamten Gruppe teilen; diese ist also mindestens gleich dem kleinsten gemeinsamen Vielfachen der $n_i$.



Experimente zweier Wissenschaftler von Bell Northern Research in Ottawa zeigten, daß die Ordnung der erzeugten Untergruppe größer als $0{,}9 \times 10^{2499}$ sein muß; das liegt sehr deutlich über $2^{56}$ liegt. Für Einzelheiten sei auf ihre Arbeit

KEITH W. CAMPBELL, MICHAEL J. WIENER: DES is not a Group, *Crypto '92, Springer Lecture Notes in Computer Science* **740** (1993), 512–520

verwiesen. Sie zeigt insbesondere, daß mehrfache Anwendung von DES die Sicherheit erhöhen kann.

### b) Doppelter DES

Beim doppelten DES hat man dann, wie bereits erwähnt, einen Schlüsselraum mit $2^{112}$ Elementen. Leider muß man diesen aber, falls man genügend viel Speicherplatz hat, nicht vollständig durchsuchen, um die Schlüssel zu finden: Falls man ein Paar aus einander entsprechendem Klartext und Chiffretext hat, reicht es, in einer sogenannten *meet in the middle attack* den Klartext mit allen $2^{56}$ möglichen Schlüsseln zu *ver*schlüsseln und den Chiffretext mit allen $2^{56}$ Schlüsseln zu *ent*schlüsseln; sobald man einen Block gefunden hat, der bei beiden Operationen auftritt, hat man mit an Sicherheit grenzender Wahrscheinlichkeit die beiden Schlüssel gefunden.

In der Praxis würde man wohl den beträchtlichen Speicheraufwand für $2^{56}$ Blöcke scheuen und versuchen, ihn in einem *time-memory-tradeoff* auf Kosten einer höheren Rechenzeit zu kompensieren; trotzdem zeigt die grundsätzliche Möglichkeit einer solchen Attacke, daß der Sicherheitsvorsprung, den ein doppelter DES gegenüber dem einfachen bietet, deutlich geringer ist, als man auf den ersten Blick meint. In der Praxis spielt der doppelte DES daher keine Rolle.

### c) Dreifacher DES

Der nächste Schritt zu Erhöhung der Komplexität besteht in einer dreifachen Anwendung von DES. Auch hier kann man wieder eine *meet in the middle attack* anwenden, aber auf dem Weg zur Mitte muß von mindestens einer der beiden Seiten aus DES zweimal mit verschiedenen