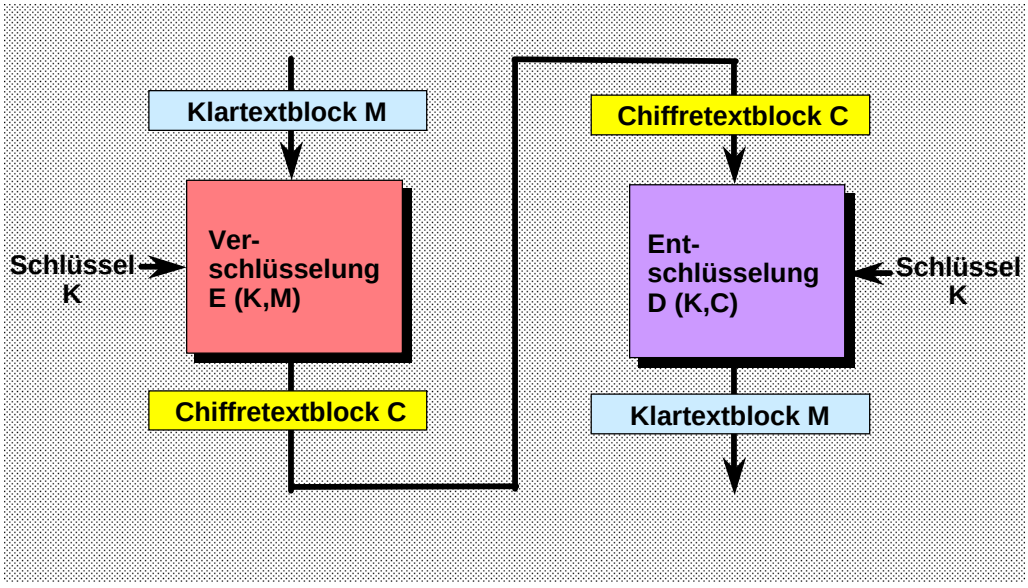
Symmetrische Verfahren

Lorenz Müller
BFH / HTI
Quellgasse 21
2505 Biel

Symmetrische Verfahren

Symmetrische Verfahren

1. **Grundprinzipien von Blockchiffren**
    1. **Produktchiffren, iterative Chiffres**
    2. **Feistel-Prinzip**
2. **Beispiele von Blockchiffren**
    1. **DES (DES3)**
    2. **IDEA**
    3. **AES**
3. **Auswahl und Einsatz von Blockchiffren**
    1. **Angriffe und Gegenmassnahmen**
    2. **Betriebsmoden**
4. **Zufallsfolgen**
    1. **Pseudozufallsgenerator**
    2. **Zufälligkeit endlicher Folgen**
5. **Lineare Schieberegister**
    1. **Rückkoppelung**
    2. **M-Folgen**
6. **Schieberegister**
    1. **Berlekamp-Massey Algorithmus**
    2. **Lineare Komplexität**
7. **Stromchiffren**
    1. **Taktgesteuerte Schieberegister**
    2. **A5**
    3. **RC4**



ALG 1-2

Grundprinzipien von Blockchiffren

Zur Verschlüsselung mit einer Blockchiffre wird eine Nachricht in Abschnitte fester Länge, sogenannte Blöcke, zerlegt. Die Blöcke werden dann nacheinander mit einem gleichbleibenden Verfahren verschlüsselt.

Die Stärke einer Blockchiffre beruht darauf, dass die Zuordnung zwischen Klartext-blöcken und Chiffretextblöcken hinreichend komplex ist. Insbesondere darf es nicht möglich sein, aus einer Anzahl zusammengehöriger Klartext-/Chiffretextblöcke den verwendeten Schlüssel zu berechnen (*known-plaintext* Angriff).

Um überhaupt genügend Variabilität für die Verschlüsselung zu ermöglichen, müssen die Blöcke eine gewisse Mindestgrösse (in der Regel mindestens 64 bit) aufweisen.

Im folgenden bezeichnet M einen Klartextblock, C einen Chiffretextblock, K einen Schlüssel. **E**(.,.) steht für die Verschlüsselungsoperation und **D**(.,.) für die Entschlüsselungsoperation einer Blockchiffre.

Es gilt:

$$C = \mathbf{E}(K,M), \quad M = \mathbf{D}(K,C)$$

Jeder Klartextblock muss aus dem zugehörigen Chiffretextblock eindeutig zurückgewonnen werden können, d.h. die Abbildung E(K,.) muss für jeden Schlüssel K injektiv sein. Aus diesem Grund sind die Chiffretextblöcke zwangsläufig mindestens so gross wie die Klartextblöcke. Sind sie grösser, so übersteigt die Länge des Chiffretextes die Länge des Klartextes. Eine derartige Expansion ist im allgemeinen unerwünscht. Daher stimmt die Länge der Klar- und Chiffretextblöcke in den meisten Blockchiffren überein. Die betrachteten Abbildungen müssen daher notwendig bijektiv, also Permutationen auf der Menge der Blöcke sein.

Blocklänge	Anzahl Blöcke	Anz. bijektiver Funktionen	Theoretische Schlüssellänge
n	2^n	$2^n!$	$\log2(2^n!)$
1	2	2	1
2	4	24	5
4	16	$\approx 2.1\ 10^{13}$	45
8	256	$\approx 8.6\ 10^{506}$	1‘684
32	$\approx 4.3\ 10^9$	...	$\approx 10^{11}$
64	$\approx 1.8\ 10^{19}$	...	$\approx 10^{21}$

ALG 1-3

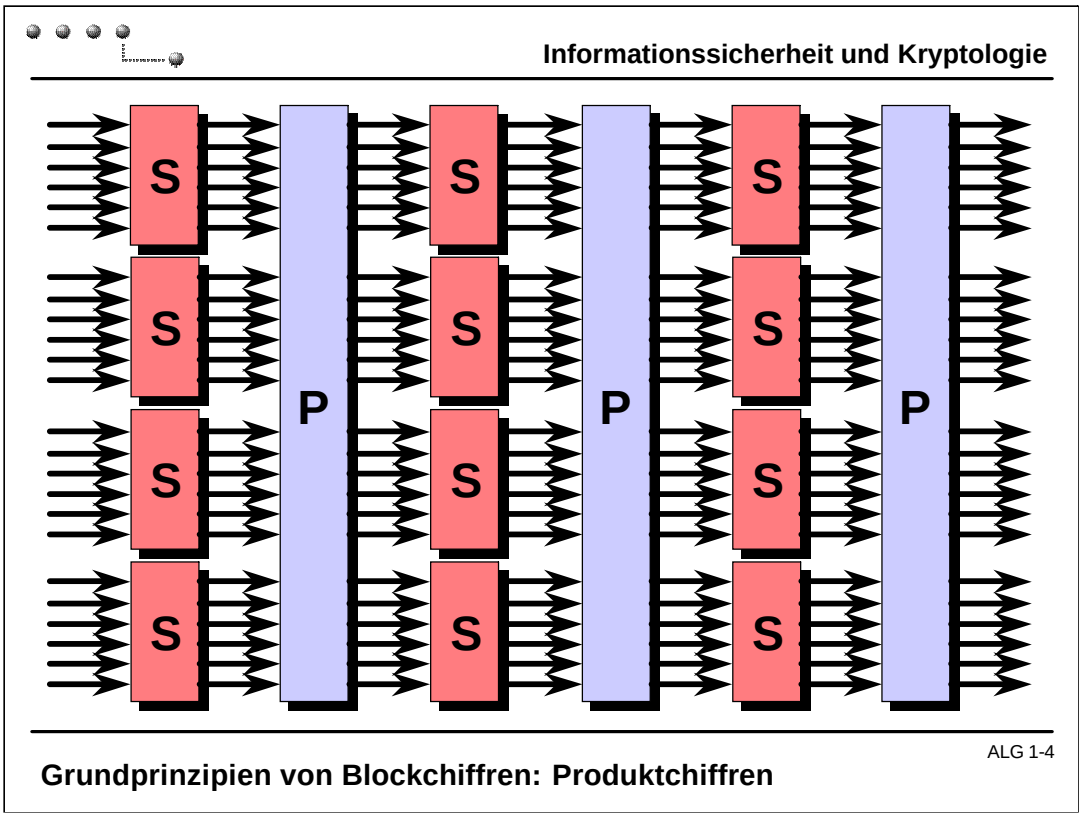
Grundprinzipien von Blockchiffren: Schlüsselraum

Bei einer Blocklänge n gibt es 2^n verschiedene Blöcke und damit $2^n!$ verschiedene Permutationen. Um jeweils eine dieser Permutationen auszuwählen, benötigt man eine Schlüssellänge von $\log_2(2^n!)$ Bit. Für die (in der Praxis natürlich völlig unzu-reichende) Blocklänge 4 ergäbe sich eine Schlüssellänge von 45 Bit. Für die Block-länge 8 würden bereits 1684 Schlüsselbits benötigt.

Um alle möglichen Verschlüsselungen eines 64 Bit langen Blocks darzustellen, müsste der Schlüssel sogar ca. 10^{21} Bit lang sein.

Aus praktischen Gründen (Schlüsselmanagement) muss man sich üblicherweise mit einer viel geringeren Schlüssellänge in der Grössenordnung von 64, 128, 196 oder 256 Bit zufrieden geben.

Die Schwierigkeit beim Entwurf der Blockchiffre besteht daher im wesentlichen darin, eine für kryptographische Zwecke geeignete (sehr) kleine Teilmenge aller möglichen Permutationen zu finden, die auch noch mit möglichst geringem Aufwand und effizient implementiert werden können.



Die in der Praxis möglichen Arten von Verschlüsselungsverfahren sind vor allem dadurch eingeschränkt, dass sie effizient realsierbar sein müssen.

Funktionen, bei denen der Output auf sehr komplexe Weise vom Input abhängt, werden oft durch Tabellen implementiert. Schon bei mässiger Blockgrösse wirft eine Tabellen-implementierung allerdings unlösbare Speicherprobleme auf. Bei n Bit Blockgrösse belegt die Tabelle $n2^n$ Bit Speicher. Für n = 16 ergibt dies ca. 1 Megabit, für n = 32 bereits mehr als 100 Gigabit und für n = 64 würde man einen über 10^{12} Gigabit grossen Speicher benötigen.

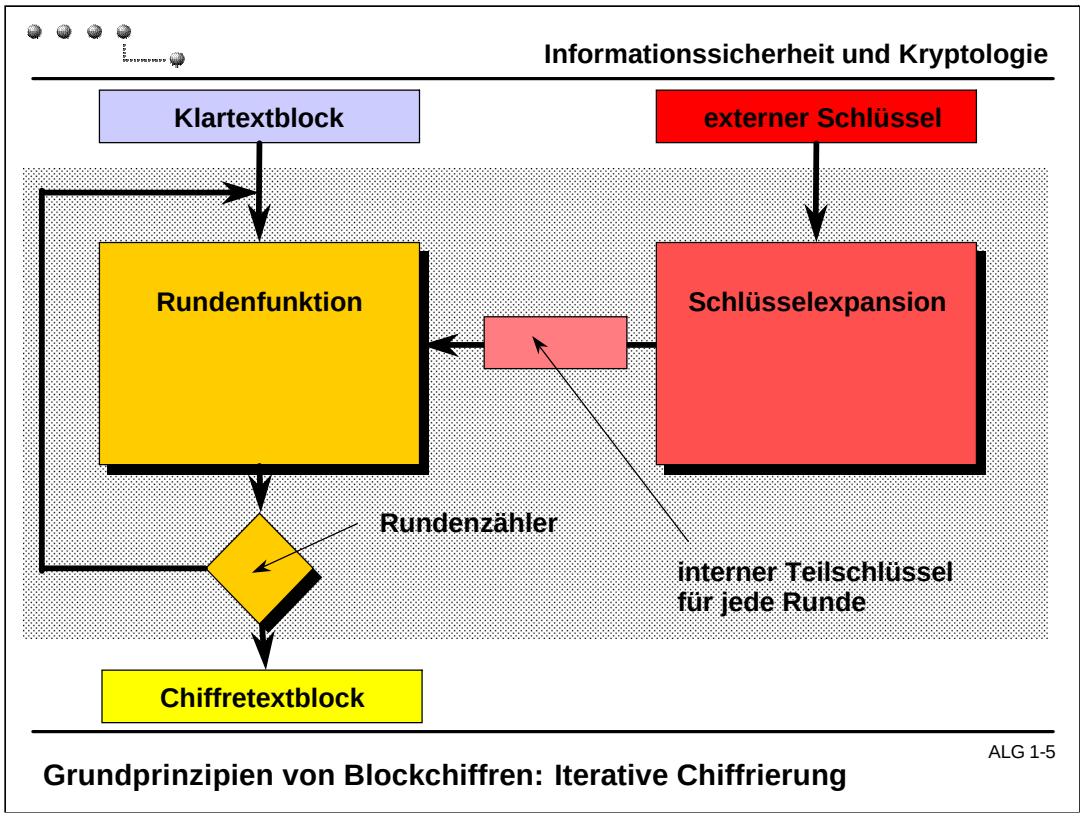
Für Blockgrössen von üblicherweise 64 oder 128 Bit werden daher Substitutionen benötigt, die so aufgebaut sind, dass sie wesentlich effizienter implementiert werden können.

Eine prinzipielle Möglichkeit dafür besteht darin, die Chiffre als sogenannte Produkt-chiffre mit Hilfe von mehreren kleineren Substitutionen (die häufig als S-Boxen bezeichnet werden) und Permutationen (P-Boxen) zu realisieren.

Durch die abwechselnde Iteration von Substitutionen und Permutationen kann man, eine hinreichen komplexe Verschlüsselungsoperation erhalten.

Übungsaufgabe:

Überlegen Sie, welche Eigenschaften die P-Boxen haben sollten, damit bereits nach möglichst wenigen Durchgängen jedes der resultierenden Bits von jedem Inputbit abhängt.



Grundprinzipien von Blockchiffren: Iterative Chiffrierung

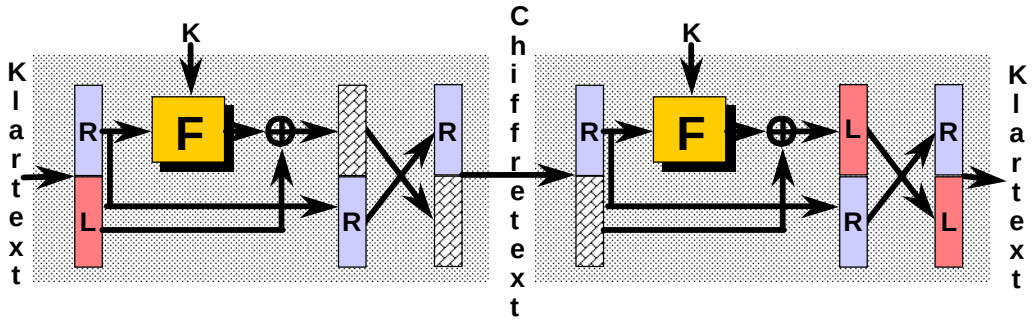
Um die Implementierung einer Produktchiffre zu vereinfachen, wird man sich in der Praxis darauf beschränken, in jeder Schicht dieselben Substitutionen und Permutationen einzusetzen und lediglich den jeweils gebrauchten Schlüssel zu variieren.

Die Chiffrierung kann dann als ein iterativer Vorgang beschrieben werden, dessen einzelne Iterationen als Runden bezeichnet werden. In jeder Runde wird ein neuer, sogenannter interner Teilschlüssel aus dem externen Schlüssel gebildet.

Die Sicherheit der Chiffre hängt dann aussert von der geschickten Auswahl der Runden-funktion auch von der Anzahl der Runden und der Art und Weise der Schlüsselexpansion ab.

Übungsaufgabe:

Überlegen Sie, in welcher Weise eine ungeeignete Schlüsselexpansion die Sicherheit der Verschlüsselung beeinträchtigen könnte.



$$C_L = (L \oplus F(R)) \qquad L = (C_L \oplus F(R)) = (L \oplus F(R) \oplus F(R))$$

$$C_R = R \qquad R = C_R = R$$

ALG 1-6

Grundprinzipien von Blockchiffren: Feistel-Prinzip

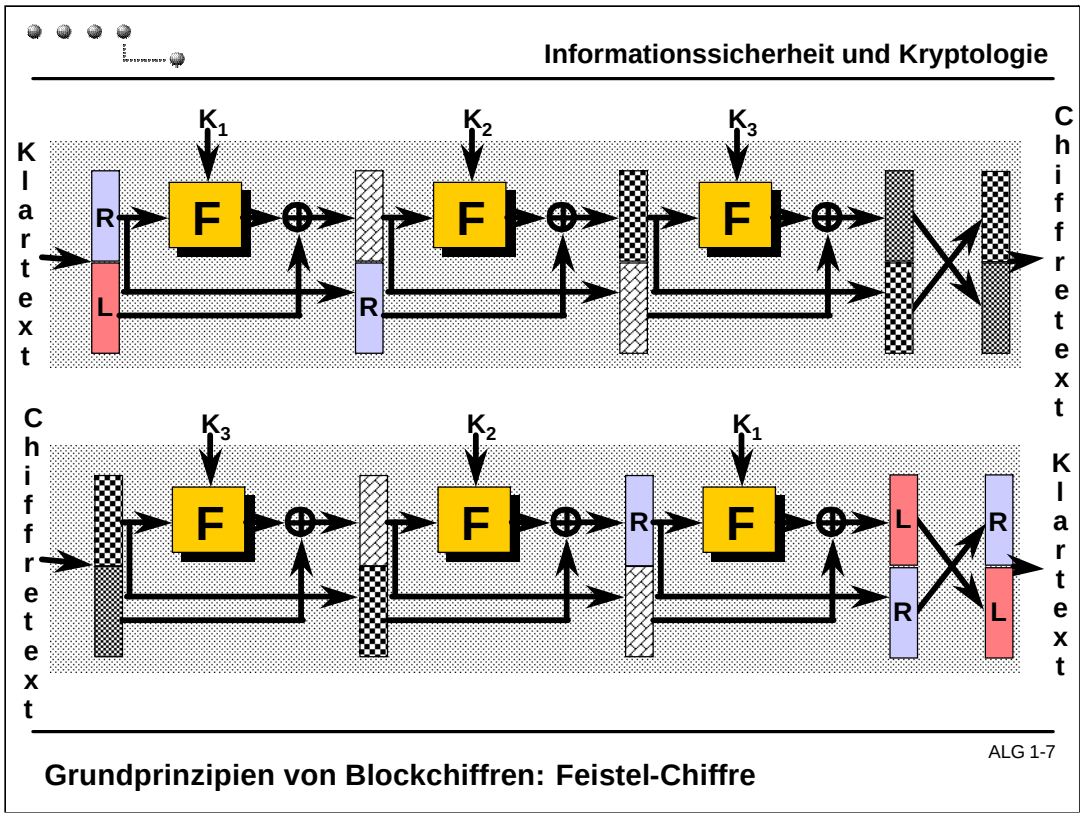
Eine Klasse von Blockchiffren mit einer besonderen Art von Rundenfunktion wird nach H. Feistel benannt.

In der obigen Abbildung wird der Klartextblock M in zwei gleich grosse Teilblöcke L und R zerlegt. Der Chiffretextblock wird dann aus dem Teilblock R und der bitweisen XOR-Verknüpfung des Teilblocks L mit dem Funktionswert F(K,R) zusammengesetzt. Wenn die beiden Hälften nun noch einmal vertauscht werden, entsteht daraus eine selbstinverse Funktion, da

F(K,R) + (F(K,R) + L) = L

gilt.

Die kryptographische Stärke der Chiffre hängt dabei von der (nicht notwendig umkehrbaren) Funktion F ab. Da aber eine Hälfte des Klartextes vollkommen unverschlüsselt bleibt, taugt das in der Abbildung gezeigte Prinzip noch nicht als Chiffre, wohl aber als Rundenfunktion einer iterativen Chiffrierung.



Die Abbildung zeigt die Ver- und Entschlüsselung einer Feistel-Chiffre mit drei Runden. Zur Dechiffrierung müssen die Teilschlüssel in genau der umgekehrten Reihenfolge angewendet werden wie bei der Chiffrierung. Die zusätzliche Vertauschung der Teilblöcke ist nur am Ende der dritten Runde erforderlich.

Übungsaufgabe:

Eine Blockchiffre nennt man ‚vollständig', wenn jedes Bit des Chiffretexts von jedem Bit des Klartextes abhängt.

Überzeugen Sie sich, dass eine Feistel-Chiffre mit drei Runden vollständig ist, falls die Funktion F vollständig ist, und dass zwei Runden dazu noch nicht genügen.

Data Encryption Standard (DES)

- **Schlüssellänge 56 Bit**
- **Seit 1976 Standard**
- **Sehr grosse Verbreitung vor allem in Bankanwendungen**

Triple-DES

- **Dreifach verschlüsseln mit zwei verschiedenen Schlüsseln (K1-K2-K1)**
- **Schlüssellänge 2*56 = 112 Bit**
- **Heute weit verbreiteter Ersatz für DES**

International Data Encryption Algorithm (IDEA)

- **1990 durch ETHZ und Ascom als Nachfolger für DES entwickelt**
- **Schlüssellänge 128 Bit**
- **Bekannt durch Einsatz im E-Mail Sicherheitsprogramm PGP**

Advanced Encryption Standard (AES)

- **Seit 1997 Auswahlverfahren für offiziellen DES-Nachfolger**
- **Kandidat wurde ausgewählt, das Verfahren ist noch nicht abgeschlossen**
- **EAS unterstützt verschiedene Block- und Schlüssellängen**

Beispiele Blockchiffren: Wichtigste Chiffren

Die auch heute noch bekannteste symmetrische Blockchiffre ist der inzwischen über 25 Jahre alte Data Encryption Standard (DES).

Bereits kurz nach der Veröffentlichung von DES wurde mit Kritik nicht gespart. Schon damals konzentrierte sich die Kritik haupt-sächlich auf die nach Meinung vieler Experten zu geringe Anzahl möglicher Schlüssel.

Heute ist DES für Anwendungen, die ein ernsthaftes Mass an Sicherheit erfordern, definitiv und anerkanntermassen nicht mehr geeignet. Eine immer häufiger genutzte Möglichkeit, die Sicherheit von DES zu erhöhen ist die dreifach iterierte Anwendung von DES (“Triple-DES”).

$$C = DES(K1, DES^{-1}(K2, DES(K1,M)))$$

Einige Zeit galt IDEA als die wichtigste Alternative zu DES. Als wesentliche Anforderungen an den Entwurf dieser Blockchiffre wurden eine im Vergleich zu DES höhere Sicherheit und eine effiziente Implementierbarkeit sowohl in Hardware, wie auch in Software gestellt.

Dies führte zum Einsatz von 128 bit langen Schlüsseln und einer ganz anderen Struktur, die wesentlich schnellere Implementierun-gen (vor allem in Software) zulässt, als sie mit DES oder gar Triple-DES möglich sind.

1997 begann ein öffentliches Auswahl-verfahren des NIST für einen Nachfolger von DES als Standard. Aus ursprünglich 15 Kandidaten wurden in einer ersten Auswahlrunde zunächst fünf Finalisten bestimmt (MARS, RC6, Rijndael, Serpent, Twofish).

Aus der zweiten Auswahlrunde ging das „Rijndael“ genannte Chiffrierverfahren als Sieger hervor.

DES (Data Encryption Standard)

FIPS PUB 46-2
Supersedes FIPS PUB 46-1
1988 January 22

Federal Information
Processing Standards Publication 46-2

1993 December 30

Announcing the Standard for

DATA ENCRYPTION STANDARD (DES)

Beispiele Blockchiffren: Geschichte von DES

Die Idee der Normung eines Verschlüsselungsverfahrens geht ursprünglich auf eine Initiative des US-Wirtschaftsministeriums aus dem Jahre 1972 zurück. Die Fachwelt wurde aufgefordert, geeignete Vorschläge einzureichen. Aufgrund der geringen Resonanz musste 1974 ein zweiter Aufruf erlassen werden.

Der einzige Vorschlag, der daraufhin in die engere Wahl kam, stammte von IBM. Es handelte sich um einen von H. Feistel entwickelten Blockchiffre namens Lucifer. Der Vorschlag wurde von der National Security Agency (NSA) geprüft und nach einer Reihe von Modifikationen 1975 offiziell als Standard vorgeschlagen.

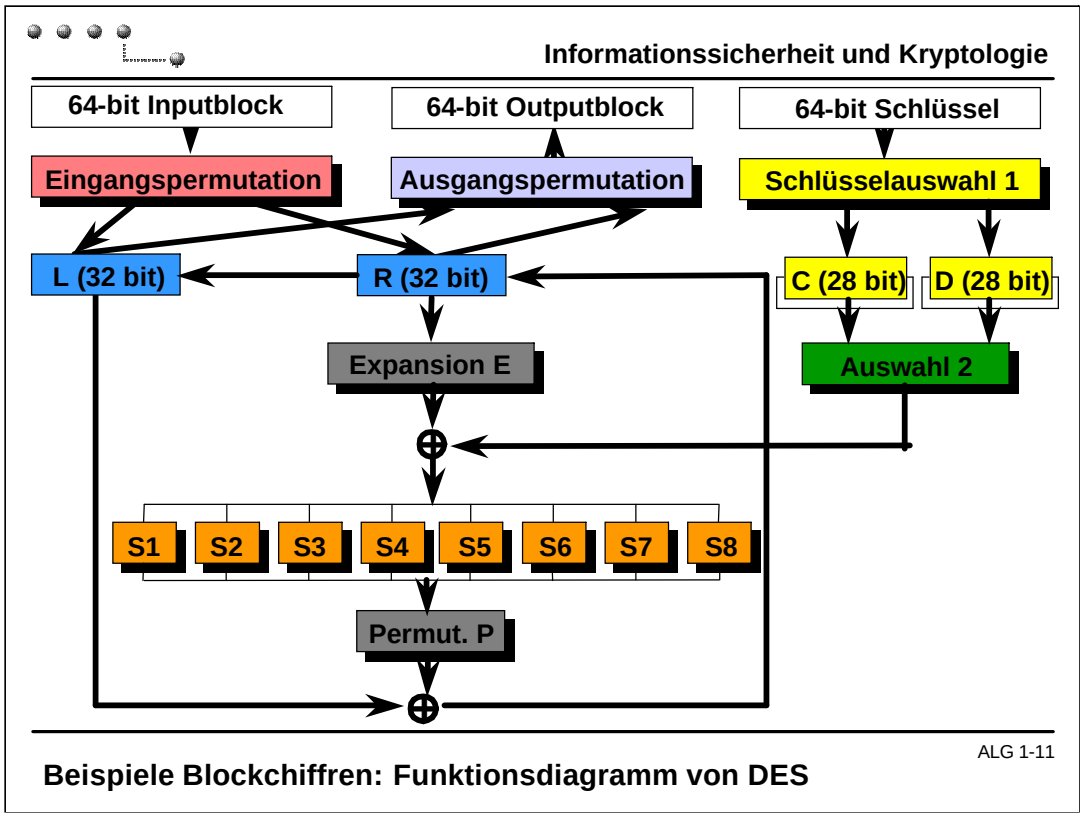
Eine der auffälligsten Modifikationen war die Reduktion der Schlüssellänge von 128 auf 56 Bit. Ausserdem wurden die Entwurfskriterien, die der Chiffre zugrunde lagen als geheim eingestuft.

Sowohl die Verkürzung der Schlüssellänge, wie auch die Geheimhaltung der Entwurf-kriterien wurden in der Folge zu Anlässen für harte Kritik an DES. Dennoch hat sich DES in der Folge durchsetzen können. Entscheidend dafür war neben dem Mangel an Alternativen sicher auch die Verfügbarkeit von schnellen und relativ preiswerten DES-Chips.

1981 wurde DES auch unter der Bezeichnung Data Encryption Algorithm (DEA) vom American National Standards Institute (ANSI) normiert. Die internationale Standardisierung durch die ISO wurde diskutiert, scheiterte schliesslich aber an grundsätzlichen Bedenken gegen die Normierung von Verschlüsselungsverfahren.

Wie jeder NIST-Standard wird DES in regelmässigen Abständen einer Überprüfung unterzogen. 1992 wurde DES letztmalig für weitere fünf Jahre bestätigt.

Der NIST-Standard enthält alle Einzelheiten der Spezifikation. Er ist als weiterführende Literatur im WebCT vorhanden. Natürlich ist DES auch in fast allen Büchern über Kryptologie ausführlich beschrieben.



Beispiele Blockchiffren: Funktionsdiagramm von DES

DES operiert auf 64-Bit Klartext- bzw. Chiffretextblöcken. Als Schlüssel verwendet DES 56 Bit, die durch acht für die Chiffrierung nicht relevante Paritätsbits auf einen 64-Bit Block ergänzt werden.

Jeder Inputblock wird zunächst einer festen Eingangspermutation unterworfen und in zwei 32-Bit Teilblöcke L und R zerlegt. Diese werden dann in 16 Runden verschlüsselt, wobei jeweils ein anderer 48-Bit Teilschlüssel in die Operation eingeht. Nach Durchführung der 16 Runden wird auf die beiden resultierenden 32-Bit Blöcke die zur Eingangspermutation inverse Ausgangspermutation angewendet.

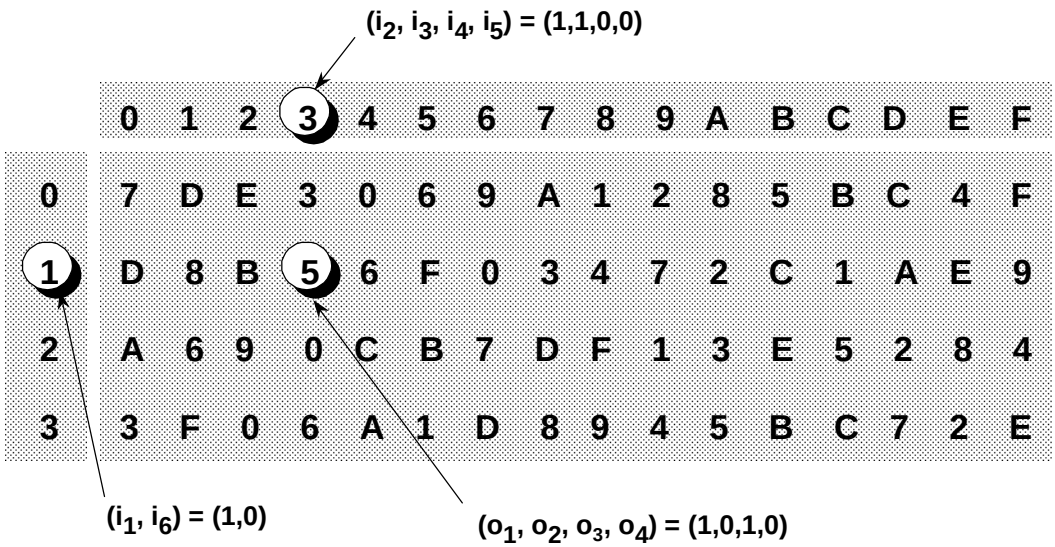
In jeder Runde wird aus dem 32-Bit Block R durch die Expansionsabbildung E ein 48-Bit Block gebildet, der komponentenweise zum 48-Bit Teilschlüssel addiert wird. Das Ergebnis wird sodann in acht 6-Bit Blöcke zerlegt, die den Input für jeweils eine der acht S-Boxen bilden. Der Output dieser Boxen ist jeweils 4 Bit (womit sie also nicht bijektiv und keine echten Substitutionen sind).

Nach Anwendung der Permutation P wird der so entstandene 32-Bit Block komponen-tenweise zu L addiert. Das Ergebnis der Operation bildet dann den neuen Block R; der alte Block R ersetzt den Block L.

Wie bereits erwähnt, werden nur 56 Bit des 64-Bit Schlüsselblocks tatsächlich ver-wendet. Die erste Schlüsselauswahl trägt diese in zwei je 28 Bit lange Schieberegister C und D ein. Die Register C und D werden vor jeder Verschlüsselungsrunde jeweils zyklisch um ein oder zwei Bit nach links geschoben. Während der 16 Runden werden insgesamt 28 Schiebeoperationen vorgenommen, so dass sich die Register nach der Verschlüsselung wieder im Ausgangszustand befinden. Die zweite Schlüsselauswahl entnimmt den Registern C und D jeweils 48 Bit für den Teilschlüssel jeder Runde

Die Dechiffrierung erfolgt in der gleichen Weise wie die Chiffrierung, wobei jedoch die Teilschlüssel in der umgekehrten Reihenfolge erzeugt werden. Dies kann dadurch erreicht werden, dass die Register C und D nunmehr rechts statt links geschoben werden.

Tabelle der S-Box S4: Input (i_1, i_2, i_3, i_4, i_5, i_6); Output (o_1, o_2, o_3, o_4)



Beispiele Blockchiffren: DES S-Boxen

Jede der acht S-Boxen substituiert einen 6-Bit Inputblock (i_1, i_2, i_3, i_4, i_5, i_6) durch einen 4-Bit Outputblock (o_1, o_2, o_3, o_4). Dabei sind die Inputbits i_1 und i_6 jeweils diejenigen, welche in der zuvor ausgeführten Expansion durch Kopieren benachbarter Bits hinzugefügt wurden. Diese werden in der jeweiligen S-Box zur Auswahl einer von vier bijektiven 4-Bit Substitutionen für den Teilblock (i_2, i_3, i_4, i_5) benutzt.

Jede S-Box kann somit durch vier Substitutionstabellen mit je 16 Einträgen à 4 Bit beschrieben werden. Die S-Box S4 ist in dieser Form in der obigen Abbildung aufgeführt. Jede Zeile der Tabelle entspricht einer 4-Bit Substitution; Input und Output sind hexadezimal angegeben. Mit dem Inputblock (1,1,1,0,0,0) erhält man also beispielsweise den in der Zeile (1,0) = 1 und der Spalte (1,1,0,0) = 3 stehenden Wert 5 = (1,0,1,0).

Die Tabellen der übrigen S-Boxen sowie aller anderen Funktionsblöcke des DES kann man der Originalveröffentlichung des Standards entnehmen.

externer Schlüssel	C-Register	D-Register
01 01 01 01 01 01 01 01	0 0 0 0 0 0 0	0 0 0 0 0 0 0
FE FE FE FE FE FE FE FE	F F F F F F F	F F F F F F F
1F 1F 1F 1F 1F 1F 1F 1F	0 0 0 0 0 0 0	F F F F F F F
E0 E0 E0 E0 E0 E0 E0 E0	F F F F F F F	0 0 0 0 0 0 0

ALG 1-13

Beispiele Blockchiffren: Schwache DES-Schlüssel

In der sehr umfangreichen Literatur über DES werden zahlreiche Tests und Analysen beschrieben und eine Vielzahl von Eigenschaften dieser Chiffre diskutiert. Dabei zeigten sich schon bald einige auf den ersten Blick erstaunliche Regularitäten.

So führt die gewählte Methode der Auswahl der internen Teilschlüssel dazu, dass nicht alle der 256 möglichen externen Schlüssel die gleiche Sicherheit bieten. Ist nämlich der externe Schlüssel so beschaffen, dass die Register C und D jeweils ausschliesslich mit Nullen oder Einsen gefüllt werden, so werden zwangsläufig 16 identische interne Teilschlüssel entstehen.

Man überlegt sich leicht, dass es vier externe Schlüssel gibt, deren interne Teilschlüssel alle gleich sind. Diese werden als schwache Schlüssel bezeichnet.

Ist K ein schwacher Schlüssel, so gilt für alle Klartexte M:

$$\mathbf{DES}^2(K,M) = \mathbf{DES}\,(K, \mathbf{DES}\,(K,M)) = M$$

Darüber hinaus gibt es für jeden schwachen Schlüssel 2^{32} Fixpunkte, d.h. Klartextblöcke M, für die gilt:

$$\mathbf{DES}\,(K,M) = M$$

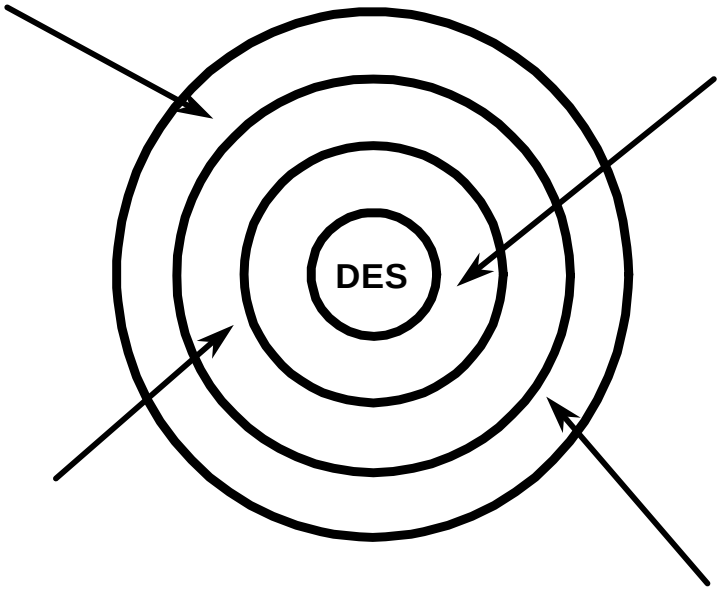
Eine weitere, zunächst unerwartete Eigenschaft von DES ist seine Invarianz gegenüber Komplementbildung. Es gilt nämlich:

$$\mathbf{DES}\,(\overline{K},\overline{M}) = \overline{\mathbf{DES}\,(K,M)}$$

wenn $\overline{}$ die bitweise Komplementbildung (d.h. jede 0 wird zu einer 1 und jede 1 zu einer 0) bezeichnet.

Übungsaufgabe:

Überlegen Sie anhand des Funktionsdiagramms von DES, warum die Invarianz gegenüber Komplementbildung gilt.



ALG 1-14

Beispiele Blockchiffren: Kryptoanalyse von DES

Bereits kurz nach der ersten Veröffentlichung von DES im Jahre 1975 wurde mit Kritik nicht gespart und es gab zahlreiche Fachleute, die von einer Standardisierung dieses Algorithmus in der vorliegenden Form dringend abrieten.

Die Kritik konzentrierte sich damals hauptsächlich auf die nach Meinung vieler Experten zu geringe Anzahl möglicher Schlüssel. In der Tat liess die Verkürzung der Schlüssel von 128 bit des ursprünglichen Designs (Lucifer) auf 56 bit darauf schliessen, dass die Stärke des Verfahrens absichtlich begrenzt wurde.

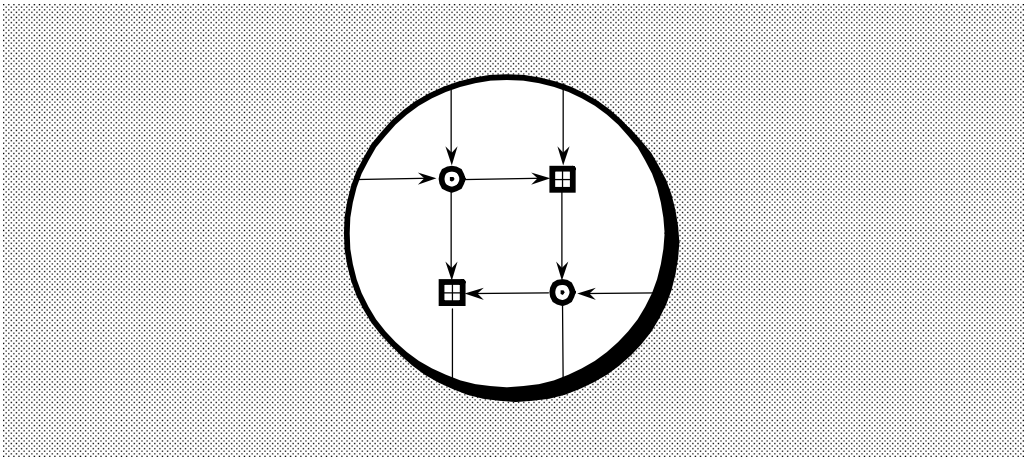
Von den vielen im Anschluss bekannt gewordenen Eigenschaften und Regulari-täten konnte allerdings längere Zeit keine für eine erfolgreiche Kryptoanalyse ausgenutzt werden. Lediglich für verkürzte Versionen von DES (d.h. mit weniger als 16 Runden) wurden einige effiziente Attacken publiziert.

Erst 1990 konnte von A. Shamir eine als differentielle Kryptoanalyse bekannt gewordene Technik angegeben werden, mit der sich DES theoretisch mit weniger Aufwand brechen lässt, als eine vollständige Schlüsselsuche erfordern würde.

Für die differentielle Kryptoanalyse, wie auch für andere inzwischen entdeckte, auf DES anwendbare Verfahren erfordern jedoch gigantische Mengen an Klartext und Chiffretext (in der Grössenordnung von 2^{50}). Daher ist die vollständige Schlüsselsuche bis heute der einzige, in der Praxis Erfolg versprechende Angriff gegen DES.

DES ist daher mit seinen 56 bit langen Schlüsseln für Anwendungen, die ein ernsthaftes Mass an Sicherheit erfordern, anerkanntermassen nicht mehr geeignet. Eine immer häufiger genutzte Möglichkeit, die Sicherheit von DES zu erhöhen ist die dreifach iterierte Anwendung von DES (“Triple-DES”). Mehr dazu folgt in Abschnitt 5.

IDEA (International Data Encryption Algorithm)



ALG 1-15

Beispiele Blockchiffren: IDEA

Bis zur Auswahl des Advanced Encryption Standard (AES) stellte der International Data Encryption Algorithm (IDEA) eine der wichtigsten Alternativen zu DES dar.

Als wesentliche Anforderungen wurden für die Entwicklung von IDEA eine im Ver-gleich zu DES höhere Sicherheit und eine effiziente Implementierbarkeit sowohl in Hardware als auch in Software gestellt.

Ein erstes Ergebnis dieser Entwicklung wurde 1990 von X. Lai und J. Massey mit dem sogenannten Proposed Encryption Standard (PES) vorgestellt. Im Jahr darauf wurde als Reaktion auf die im Sommer 1990 veröffentlichte Methode der differtentiellen Kryptoanalyse eine ursprünglich IPES (Improved PES) genannte Variante des PES präsentiert, die heute die Bezeichnung IDEA trägt.

Beim IDEA handelt es sich um eine 64 Bit Blockchiffre, die mit 128 Bit Schlüsseln arbeitet und nicht zur Klasse der Feistel-Chiffren gehört. Ihre Verschlüsselungs-operation ist aus acht Runden und einer zusätzlichen Output-Transformation zusammengesetzt.

IDEA konnte nicht als Kandidat für den AES berücksichtigt werden, da die gewählte Konstruktion nicht leicht auf andere Blocklängen als 64 Bit verallgemeinerbar ist.

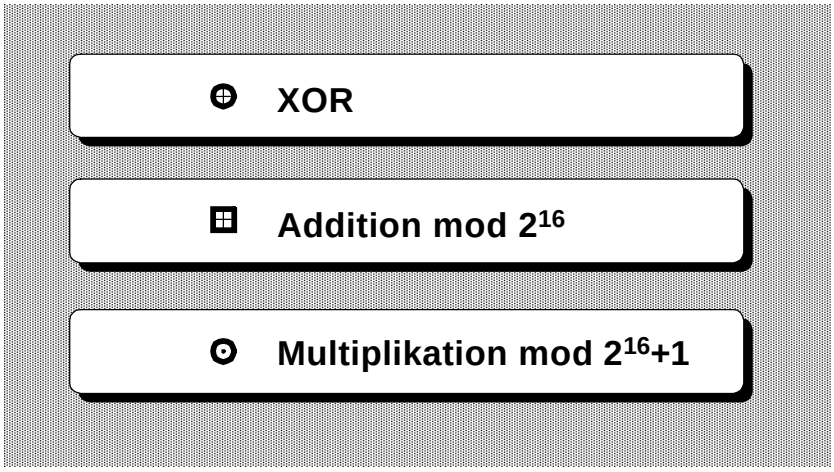
IDEA findet in der bekannten Verschlüsselungs-Software für E-Mail PGP (Pretty Good Privacy) Verwendung und wurde vor kurzem für die Verschlüsselung der grenzüberschreitenden drahtlosen Sprachkommunikation der europäischen Polizeichorps ausgewählt.

Mehr Einzelheiten zu IDEA können einem unter weiterführende Literatur im WebCT vorhanden Artikel oder den beiden in Abschnitt 6 genannten Büchern entnommen werden.

Informationen über IDEA, inkl. eines C-Programms zum Download sind zudem unter

http://www.media-crypt.com/

erhältlich.



Beispiele Blockchiffren: Entwurfsprinzip von IDEA

IDEA besitzt intern eine 16-Bit Struktur; alle ihre Grundfunktionen operieren auf 16-Bit Datenblöcken. Das Design verzichtet auf S-Boxen und Bit-Permutationen.

Bei der Verschlüsselung mit IDEA kommen lediglich drei unterschiedliche Elementaroperationen zum Einsatz:

- bitweise Addition modulo 2 (XOR)
- Addition modulo 2^{16}
- Multiplikation modulo $2^{16}+1$, wobei der Operand 0 als 2^{16} interpretiert wird

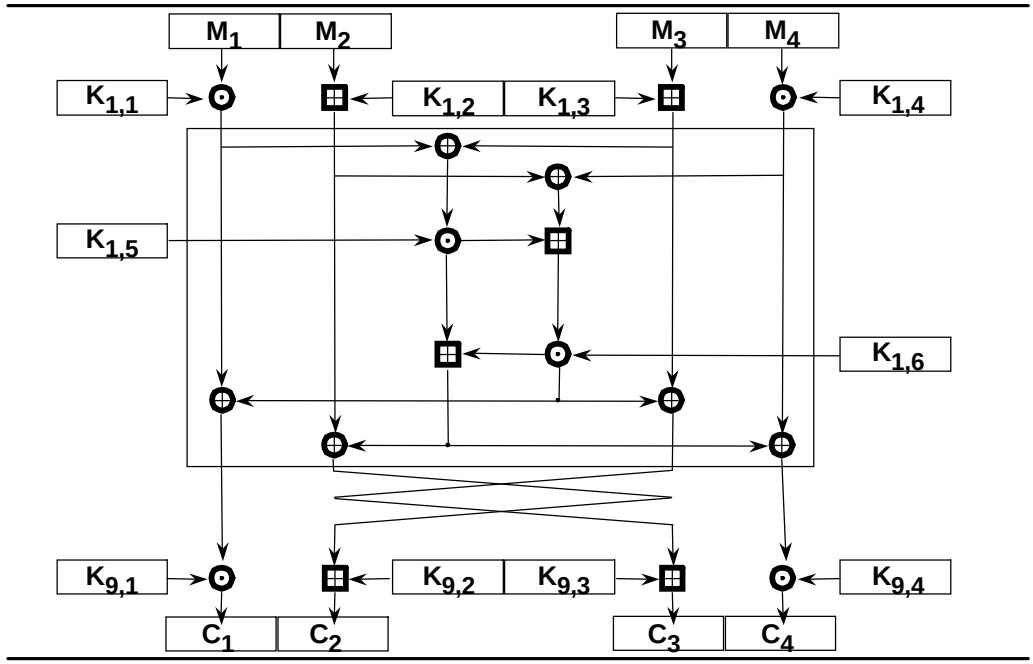
Das Grundprinzip des Entwurfs besteht darin, durch das Mischen der verschiedenen Grundoperationen bereits nach wenigen Runden eine komplexe Abhängigkeit des Chiffretexts von Klartext und Schlüssel zu erreichen. Der Output einer Grundfunktion wird dabei nie als Input für den gleichen Typ von Operation benutzt.

Die drei Operationen sind sind bewusst möglichst “inkompatibel” gewählt, d.h. sie erfüllen z.B. in keiner Kombination ein Distributiv- oder Assoziativgesetz.

Die Multiplikation modulo $2^{16}+1$ ist umkehrbar (und damit als Baustein für die Chiffrierung geeignet), weil $2^{16}+1$ eine Primzahl ist – eine sogenannte Fermat-Primzahl. Ebenso ist $2^{8}+1$ eine Primzahl, wogegen $2^{32}+1$ und $2^{64}+1$ zusammengesetzt sind.

Die Konstruktion kann kann daher unmittel-bar auf eine Blocklänge von 32 Bit (ent-sprechend vier Teilblöcken von 8 Bit) oder kürzer, leider jedoch nicht auf Blocklängen von 128 oder 256 Bit (entsprechend vier Teilblöcken von 32 bzw. 64 Bit) übertragen werden.

Dennoch ist die Skalierbarkeit auf kürzere Blocklängen eine durchaus nützliche Eigenschaft des Designs. Damit können 8, 16 oder 32-Bit Varianten definiert und analysiert werden. Verschiedene Charakteristika dieser Mini-IDEA-Chiffren wurden analysiert und erlauben Rückschlüsse auf Eigenschaften der 64-Bit Chiffre.



ALG 1-17

Beispiele Blockchiffren: Funktionsdiagramm von IDEA

Die Abbildung zeigt die prinzipielle Funktionsweise von IDEA. Dargestellt ist die erste Runde und die Output-Transformation.

Ein 64-Bit Klartextblock M wird zunächst in vier 16-Bit Teilblöcke M_1, ..., M_4 zerlegt. Diese Teilblöcke werden in acht identischen Verschlüsselungsschritten verarbeitet, wobei in jeder Runde sechs 16-Bit Teilschlüssel $K_{r,1}$, ..., $K_{r,6}$ (für Runde r) benutzt werden.

Abschliessend wird eine Output-Transformation durchgeführt, die von weiteren vier Teilschlüsseln $K_{9,1}$, ..., $K_{9,4}$ abhängt. Somit werden zur Berechnung eines Chiffretextblocks C_1, ..., C_4 insgesamt 52 interne Teilschlüssel benötigt.

Das Design der Chiffre ermöglicht es, die Entschlüsselungsoperation nach demselben Schema wie die Verschlüsselung durchzuführen. Allerdings ist zur Entschlüsselung ein anderer Satz von Teilschlüsseln erforderlich. Die Umkehrung der Verschlüsselung wird durch die Tatsache erleichtert, dass der in der Abbildung gestrichelt umrandete “Kern” der Chiffre selbstinvers ist.

Die weiteren arithmetischen Operationen sind ebenfalls umkehrbar, da jeder Teilschlüssel ein additives oder multiplikatives Inverses besitzt.

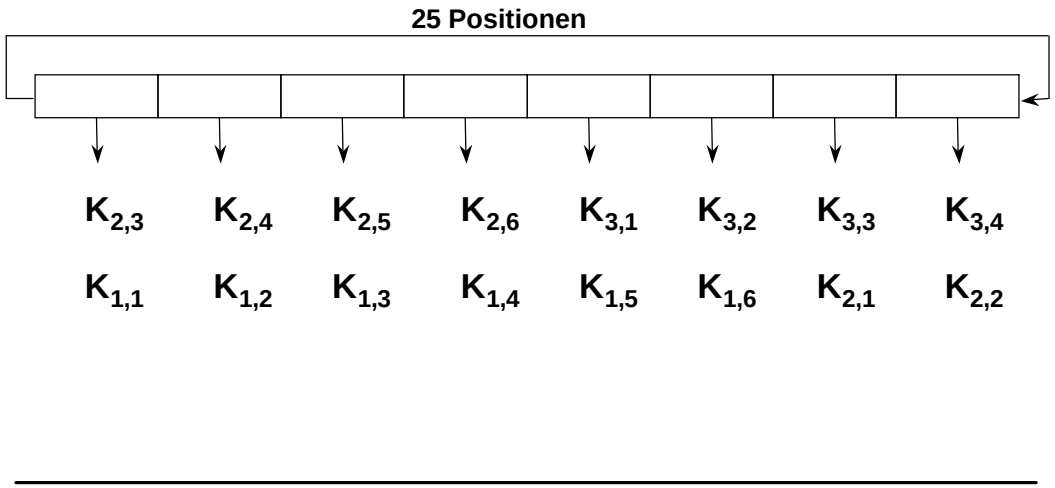
Für die Dechiffrierung sind daher die Teilschlüssel $K_{r,1}$ und $K_{r,4}$ (für r = 1, ..., 9) bezüglich der Multiplikation mod $2^{16}+1$ und die Teilschlüssel $K_{r,2}$ und $K_{r,3}$ (für r = 1, ..., 9) bezüglich der Addition mod 2^{16} zu invertieren und im Verschlüsselungsschritt 10-r anzuwenden. Die Teilschlüssel $K_{r,5}$ und $K_{r,6}$ (für r = 1, ..., 8) dagegen werden unverändert in Runde 9-r eingesetzt.

Bereits nach einer Runde ist die Chiffre bezüglich der Inputbits, nach zwei Runden auch bezüglich der Schlüsselbits vollständig.

⊕ XOR

⊞ Addition mod 2^{16}

⊙ Multiplikation mod $2^{16}+1$



ALG 1-18

Beispiele Blockchiffren: IDEA-Schlüsselauswahl

Die internen Teilschlüssel werden nach folgendem Verfahren aus dem externen 128-Bit Schlüssel K generiert. Die ersten acht Teilschlüssel $K_{1,1}$, $K_{1,2}$, ..., $K_{2,2}$ werden unmittelbar durch Zerlegen von K in acht 16-Bit Blöcke gewonnen. Für jeden weiteren Satz von acht Teilschlüsseln wird der externe Schlüssel K zunächst einem zyklischen Linksshift um 25 Bitpositionen unterzogen und dann wie oben zerlegt.

0 wirkt sich weder bei der bitweisen Addition modulo 2 noch bei der Addition modulo 2^{16} aus.

1 hat bei der Multiplikation modulo $2^{16}+1$ keinen Effekt.

0 wird bezüglich der Multiplikation modulo $2^{16}+1$ als 2^{16} (= -1) interpretiert und ist daher selbstinvers.

ALG 1-19

Beispiele Blockchiffren: Schwache IDEA-Schlüssel

Teilschlüssel, die den Wert 0 oder 1 besitzen, spielen bei den Grundoperationen von IDEA eine besondere Rolle:

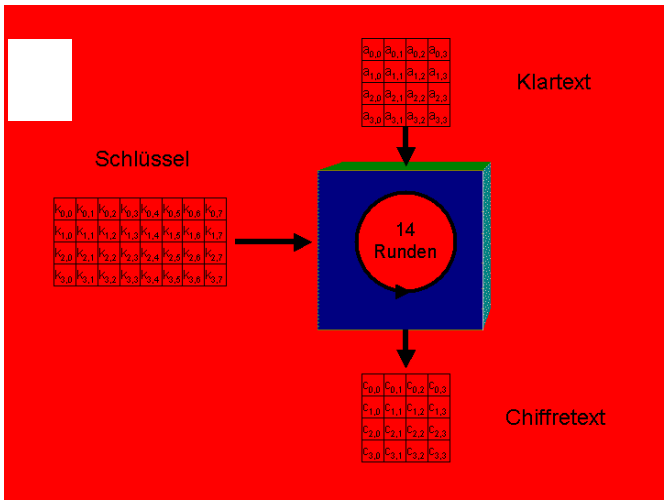
- Ein Operand 0 wirkt sich weder bei der bitweisen Addition noch bei der Addition modulo 2^{16} aus.
- Ein Operand 1 hat bei der Multiplikation modulo $2^{16}+1$ keinen Effekt.
- Der Operand 0 wird bezüglich der Multiplikation modulo $2^{16}+1$ als 2^{16} (= -1) interpretiert und ist daher selbstinvers.

Auf Grund dieser Eigenschaften werden diejenigen 16-Bit Teilschlüssel, die binär 0 oder 1 entsprechen, als schwach bezeichnet.

Es gilt insbesondere, dass die gesamte Verschlüsselung selbstinvers ist, falls man als externen Schlüssel den Nullvektor wählt. Für andere externe Schlüssel, die zu vielen schwachen Teilschlüsseln führen, sind negative Auswirkungen auf die Stärke der Chiffre weniger offensichtlich, aber vorhanden.

Grundsätzlich ist es daher ratsam, bei der Auswahl von Schlüsseln für IDEA solche mit sehr geringem Gewicht oder mit längeren Abschnitten (>15 bit) von Nullen zu vermeiden. Bei einer zufälligen Auswahl von Schlüsseln, treten solche unerwünschten Exemplare allerdings ohnehin mit einer verschwindend geringen Häufigkeit auf.

AES (Advanced Encryption Standard)



Beispiele Blockchiffren: AES (Advanced Encryption Standard)

Um einen Nachfolger von DES zu bestimmen, führte NIST ein öffentliches Auswahlverfahren durch. Aus ursprünglich 15 Kandidaten ging der von den beiden belgischen Kryptologen Joan Daemen und Vincent Rijmen entwickelte Algorithmus Rijndael als Empfehlung für den Advanced Encryption Standard AES hervor.

Derzeit läuft der mehrere Monate dauernde, formale Prozess (mit Vernehmlassung, Einspruchsfristen etc.), an dessen Ende voraussichtlich Rijndael (oder ein unwesentlich modifizierter Algorithmus) definitiv als der neue AES bestimmt werden wird.

Informationen über den Stand und den Fortgang des Verfahrens sind unter

http://csrc.nist.gov/encryption/aes/

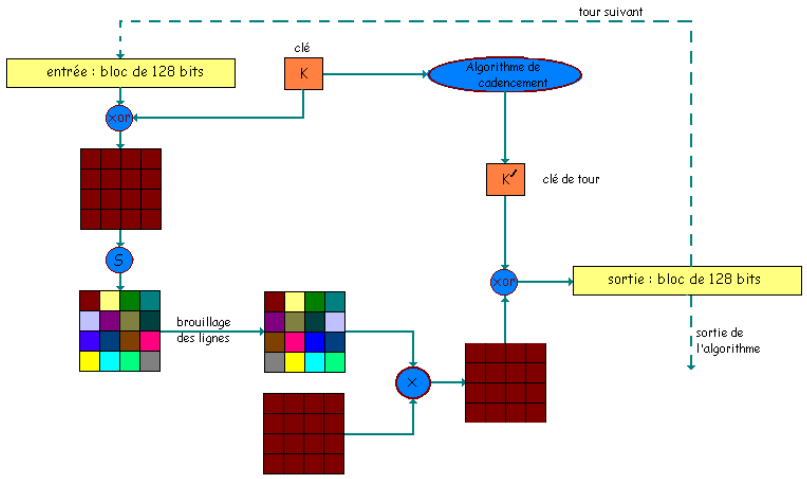
zu erhalten (siehe auch unter Links im WebCT).

Ein Überblick über die Funktionsweise von Rijndael wird im Rahmen dieses Moduls in einem separaten Online Tutorial zur Verfügung gestellt, dass während der Selbststudienphase durchgearbeitet werden kann.

Details der Spezifikation, Überlegungen zur Implementierung und eine Analyse zu Rijndael finden sich in der unter der weiterführenden Literatur um WebCT vorhandenen Originalveröffentlichung:

J. Daemen, V. Rijmen: AES-Proposal: Rijndael, Version 2, 03/09/99

AES (Advanced Encryption Standard)

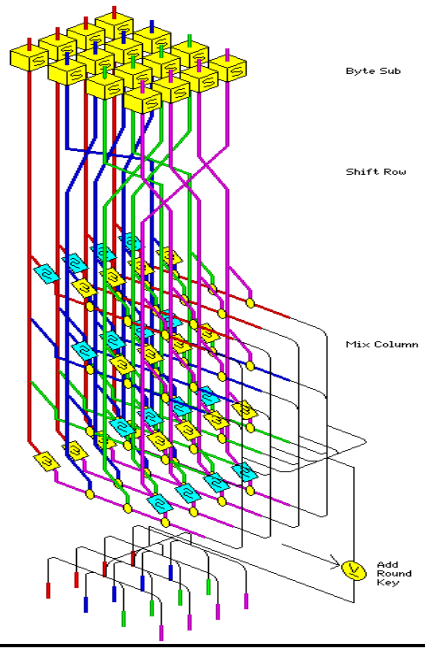


ALG 1-21

Beispiele Blockchiffren: AES (Advanced Encryption Standard)

Der Grundalgorithmus Rijndael operiert auf Blöcken von 128 bits, mit einem Schlüssel von 128 bit,.es gibt jedoch die Option für längere Blöcke und Schlüssel. Jeder Block durchläuft 10 Runden des Basisalgorithmus:

1. Addition des geheimen Schlüssels (expandiert) mit EXOR
2. Nichtlineare Transformation der einzelnen Bytes: Die 128 bits werden in eine 4 x 4 Matrix mit je 8 bits (Octet) eingeteil. Jedes Octet wird mit einer nichtlinearen Substitution transformiert.
3. Zeilenverschiebung: Die 3 unteren Zeilen der 4x4 Matrix werden zyklisch nach links rotiert. Die 2 Zeile um eine Position, die 3. Zeile um2 und die 4. Zeile um 3 Positionen..
4. Spaltendiffusion durch Multiplikation: Jede Kolonne in der Matrix wird wird mit einer 4x4 Matrix von Octeten multipliziert, was zu einer Linearkombination der einzelnen Elemente der Kolonne führt. Die Berechnungen erfolgen im Galoisfeld GF(2^8) (Addition: Bitweises EXOR; Multiplikation: Octet als formales Polynom mit Grad 7, Multiplikation als Polynommultiplikation mod ein irreduzibles Polynom m(x)=x^8 + x^4 + x^3 + x +1
5. Addition des Rundenschlüssels: Bei jeder Runde wird ein Rundenschlüssel aus dem geheimen Schlüssel genererier, welcher mit EXOR mit dem letzten Block verknüpft wird.

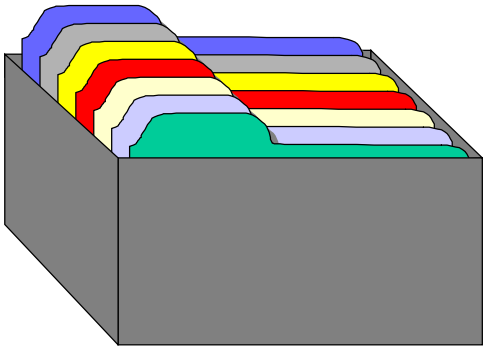


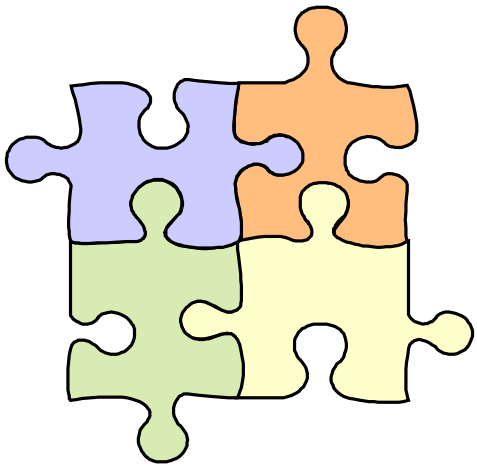
Beispiele Blockchiffren: AES (Advanced Encryption Standard)

ALG 1-22

Graphische Darstellung der AES Rundenoperation ausgehend von der 4x4 Matrix der Octets.

Auswahl und Einsatz von Blockchiffren





ALG 1-23

Auswahl und Einsatz von Blockchiffren

Die Sicherheit einer (Block)chiffre zu beurteilen, ist eine sehr aufwendige und schwierige Aufgabe für Spezialisten. Der Anwender muss sich bei der Auswahl eines geeigneten Algorithmus daher auf die Fähigkeit und Vertrauenswürdigkeit von Experten verlassen.

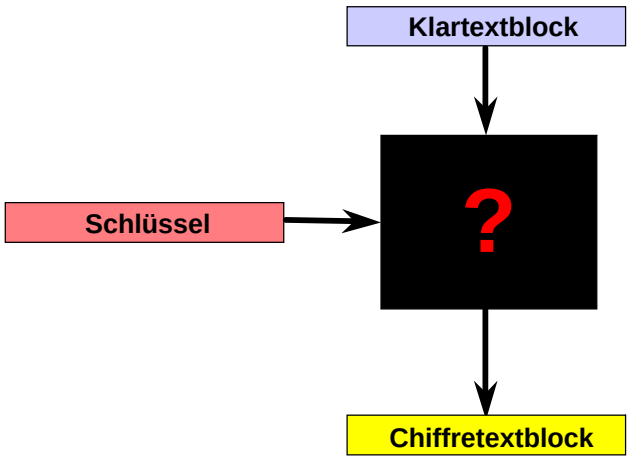
Bei der Auswahl beschränkt man sich daher mit Vorteil auf diejenigen Algorithmen, die in allen Details veröffentlicht und von vielen Kryptologen analysiert wurden.

Selbst dann ist die Auswahl eines geeigneten Verfahrens und der korrekte Einsatz für den angestrebten Zweck nicht immer einfach.

Bis vor einiger Zeit, war die Auswahl kryptographischer Algorithmen auch durch die restriktive Handhabung von Exportlizenzen durch die USA eingeschränkt. Dies hat sich inzwischen zum Besseren geändert. Dennoch ist der Einsatz von kryptographischen Verfahren in vielen Ländern durch gesetzliche Regelungen beschränkt.

Eine Übersicht der geltenden Regelungen findet sich im Internet und der folgen URL:

http://cwis.kub.nl/~frw/people/koops/lawsurvy.htm



ALG 1-24

Auswahl und Einsatz: Black-Box-Angriffe

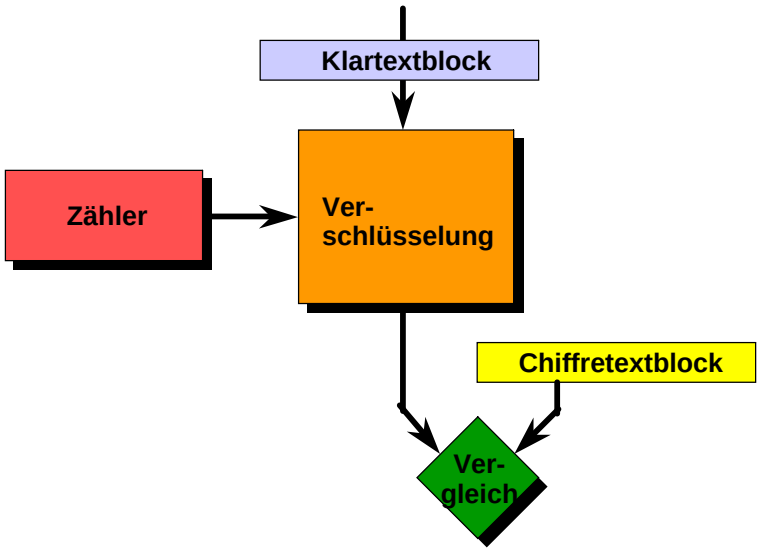
Angriffe auf eine Blockchiffre können entweder spezifische Schwachstellen der jeweiligen Konstruktion ausnutzen oder auch unabhängig von deren interner Struktur sein. Die letzteren werden auch als Black-Box-Angriffe bezeichnet und erlauben die Aufstellung genereller Mindestanforderun-gen an die Blockgrösse und Schlüssellänge einer Blockchiffre.

Die Codebuch-Analyse ist ein Angriff mit frei wählbarem Klartext. Dabei berechnet man zu einer möglichst grossen Anzahl von Klartextblöcken die unter Anwendung des unbekannten Schlüssels K erzeugten Chiffretextblöcke und speichert die zusammengehörigen Paare in der Art eines Codebuchs ab.

Bei geschicktem Aufbau und hinreichendem Umfang des Codebuchs wird ein Angreifer in manchen Fällen in der Lage sein, zumindest einige Blöcke eines längeren Chiffretexts zu entschlüsseln und sich von dieser Basis aus zu einer nützlichen Dechiffrierung vorarbeiten zu können.

Die Erfolgsaussichten einer Codebuch-Analyse hängen von der Blockgrösse, von der Redundanz des Klartextes und nicht zuletzt auch von der gewählten Betriebsart der Blockchiffre ab.

Während bei der Codebuch-Analyse das Entschlüsseln des Klartextes versucht wird, haben andere Angriffe das Ziel, den verwendeten Schlüssel zu bestimmen.



ALG 1-25

Auswahl und Einsatz: Vollständige Schlüsselsuche

Die vollständige Schlüsselsuche ist der theoretisch einfachste und bei starken Algorithmen in der Praxis wohl häufigste Angriff auf eine Blockchiffre. Es handelt sich dabei um einen Angriff mit bekanntem Klartext. Für ein gegebenes Paar vom Kartextblock und Chiffretextblock (M,C) wird durch systematisches Durchprobieren aller Schlüssel derjenige Schlüssel K ermittelt, für den

$$C = \mathbf{E}(K,M)$$

gilt.

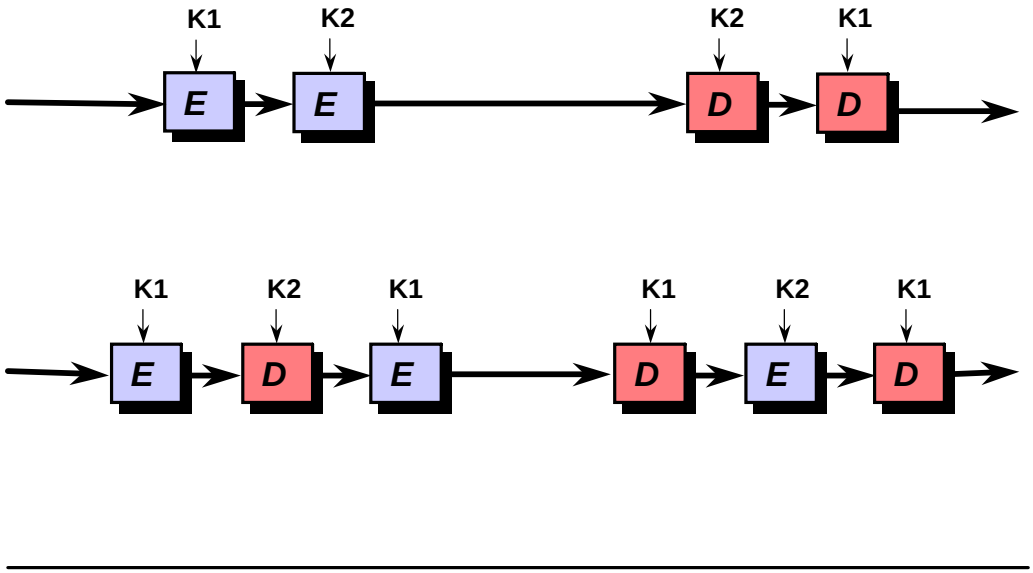
Im Mittel werden für die Suche k/2 Verschlüsselungsoperationen erforderlich sein, wenn k die Anzahl der möglichen Schlüssel ist.

Mit einem Hardware-Baustein, der pro Verschlüsselung 1µs benötigt, können in einem Jahr etwa $3 \bullet 10^{13}$ Schlüssel ausprobiert werden. Bei einem Schlüssel von 48 Bit Länge gibt es zwar schon über $2 \bullet 10^{14}$ verschiedene Schlüssel. Dennoch genügt dies für hohe Sicherheitsanforderungen kaum, da die Suche durch den Einsatz mehrerer Verschlüsselungsbausteine leicht parallelisiert werden kann.

Beim Einsatz von 1000 Bausteinen könnte im genannten Beispiel einer Chiffre mit 48 Bit langem Schlüssel etwa alle 39 Stunden ein Schlüssel ermittelt werden.

Alle praktischen Ergebnisse, die bis vor kurzem bekannt wurden, basierten auf einer massiv parallelen Schlüsselsuche mit Hilfe zahlreicher Computer, die über das Internet ihre Ergebnisse austauschen.

1998 hat die Electronic Frontier Foundation eine Maschine vorgestellt, die für $ 220’000 entwickelt und gebaut wurde und einen 56 Bit langen DES-Schlüssel im Durchschnitt in 4.5 Tagen findet.



ALG 1-26

Auswahl und Einsatz: Dreifache Verschlüsselung (Triple-DES)

Insbesondere in Zusammenhang mit der Diskussion um die Sicherheit von DES wurde die generelle Frage aufgeworfen, in wieweit man die Sicherheit (insbesondere gegen vollständige Schlüsselsuche) einer Blockchiffre durch mehrfache Hintereinander-ausführung erhöhen kann.

Entgegen der ersten Anschauung quadriert sich der Aufwand für die Schlüsselsuche keineswegs, wenn die Chiffre zweifach mit unabhängig voneinander gewählten Schlüsseln ausgeführt wird, also etwa

C = **E** (K2, **E** (K1, M)).

Statt (bei einem Angriff mit bekanntem Klartext) alle Paare von Schlüsseln zu durchsuchen, kann man nämlich einen sogenannten meet-in-the-middle Angriff ausführen.

Dabei berechnet man für alle Schlüssel (K1) die (einfachen) Chiffrierungen des bekannten Klartextblocks und für alle möglichen Schlüssel (K2) die Dechiffrierungen des zugehörigen Chiffretextblocks. Übereinstimmungen in beiden Listen, die man durch Sortieren sehr effizient bestimmen kann, ergeben Kandidaten für das tatsächlich verwendeten Schlüsselpaar.

Aus diesen Kandidaten kann man das richtige Paar durch Testen mit einem zweiten bekannten Klartext herausfiltern.

Insgesamt steigt der Aufwand für die Schlüsselsuche bei einer doppelten Verschlüsselung also nur marginal.

Eine deutliche Erhöhung ergibt sich erst bei einer dreifachen Anwendung der Chiffre (daher Tripple-DES).

Man verwendet in der Regel das Schema:

C = **E** (K1, **D** (K2, **E** (K1, M))),

da man für die Wahl K1 = K2 Kompatibilität mit der einfachen Verschlüsselung erhält. Möglich aber wenig gebräuchlich ist auch die Variante mit drei verschiedenen Schlüsseln, deren Sicherheit - wiederum wegen des meet-in-the-middle Angriffs - der Variante mit zwei Schlüsseln nur unwesentlich überlegen ist.

Der offensichtliche Nachteil der dreifachen Verschlüsselung (ob mit zwei oder drei Schlüsseln) ist der dadurch um den Faktor Drei verringerte Durchsatz.

Angreifer / Budget	40 Bit	56 Bit	64 Bit	80 Bit	90 Bit
Kleinbetrieb (ca. $10K)	Minuten	ca. 1 a	> 100 a	> 100 a	> 100 a
Grossbetrieb (ca. $10M)	Sekunden	Tage	ca. 1 a	> 100 a	> 100 a
Geheimdienst (ca. $300M)	< 1 s	Sekunden	Minuten	Jahre	> 100 a

ALG 1-27

Auswahl und Einsatz: Anforderungen an die Schlüssellänge

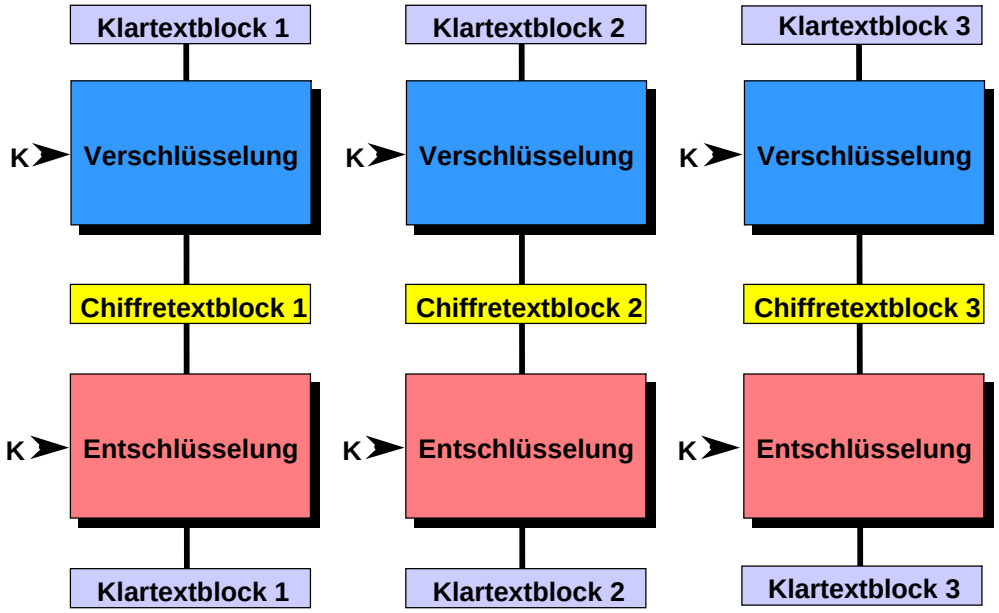
Diese Tabelle wurde auf der Basis einer 1996 veröffentlichten Studie einer Gruppe führen-der Kryptologen erstellt und mit Hilfe von Moore's Gesetz auf den heutigen Stand hochgerechnet (Moore's Gesetz besagt, dass sich die zu gegebenen Kosten verfügbare Rechenleistung etwa alle 18 Monate verdoppelt).

Die genannten Zeit- und Kostenaufwände sind als grobe Richtwerte zu verstehen und dienen nur der Bestimmung der Grössenordnung der heute und in Zukunft benötigten Schlüssellängen.

Sie basieren auf der Annahme, dass die Schlüsselsuche mit Hardware-Unterstützung erfolgt, d.h. mit dem parallelen Einsatz sehr vieler FPGA-Chips (Field Programmable Gate Array) in Fall der "Klein- und Gross-betriebe" oder ASICs (Application Specific Integrated Circuits) im Fall der "Geheim-dienste".

Aus der Tabelle ergibt sich die Faustregel, dass heute selbst bei der Konzeption von Systemen im Bereich der taktischen Sicherheit (d.h. die Informationen müssen nur relativ kurze Zeit geheimgehalten werden) 80 Bit eine untere Grenze für die Schlüssellänge darstellen sollte.

Für Systeme, die höheren Sicherheitsansprüchen genügen sollen und voraussichtlich auch in 10 Jahren und darüber hinaus noch eingesetzt werden, sind aus heutiger Sicht Schlüssel von mindestens 90 Bit Länge erforderlich. Der geschätzte Aufwand für das Brechen eines 80 Bit bzw. 90 Bit langen Schlüssels durch einen "Geheimdienst" im Jahre 2010 beläuft sich auf eine Grössenordnung von einigen Tagen bzw. mehreren Jahren.



ALG 1-28

Auswahl und Einsatz: Electronic Code Book (ECB) Mode

Die in der Abbildung dargestellte Betriebsart einer Blockchiffre wird als ECB-Modus bezeichnet. In dieser Betriebsart bildet die Chiffrierung mit einem Schlüssel K eine Folge von Klartextblöcken jeweils unab-hängig voneinander in eine Folge von Chiffretextblöcken ab.

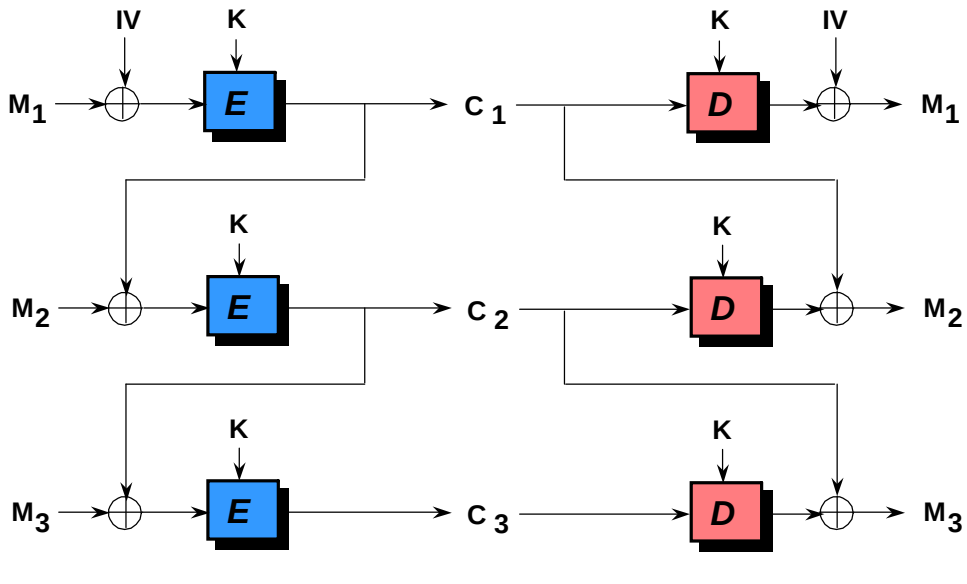
Das Problem mit dem ECB-Modus liegt in erster Linie darin, dass identische Klartextblöcke auf identische Chiffretextblöcke abgebildet werden, was in vielen Fällen eine Codebuchanalyse ermöglicht.

Die Klartext-Nachrichten sind häufig stark strukturiert, so dass bestimmte Teile sich nicht selten sogar innerhalb einer einzelnen Nachricht mehrfach wiederholen. Auch weisen verschiedene Nachrichten häufig gemeinsame Teile auf (z.B. Floskeln am Anfang und Ende). Nachrichten, die von Computerprogrammen erzeugt werden (z.B. Zahlungsanweisungen) sind oft besonders stark strukturiert.

Neben der Gefahr der Codebuchanalyse weist der ECB Modus eine weitere Schwachstelle auf, nämlich die Möglichkeit, einzelne Chiffretextblöcke sehr einfach und unbemerkt zu löschen, einzufügen und zu vertauschen.

Daher sollte ECB eigentlich niemals zur Verschlüsselung von Nachrichten gebraucht werden, die länger als ein einzelner Block sind.

Aber auch mit kurzen Nachrichten gibt es ein Problem. Wegen der Gefahr der Codebuchanalyse darf ein unvollständiger Klartextblock nicht immer mit demselben Bitmuster (etwa lauter Nullen) aufgefüllt werden, sondern sollte mit einem zufälligen Bitmuster ergänzt werden.



ALG 1-29

Auswahl und Einsatz: Cipher Block Chaining (CBC) Mode

Eine bessere Methode, mehrere zusammen-gehörige Blöcke zu verschlüsseln, ist der sogenannte Cipher Block Chaining Mode. Der wesentliche Vorteil dieser Betriebsart ist, dass gleiche Klartext-Blöcke nicht automatisch auf gleiche Chiffretext-Blöcke abgebildet werden.

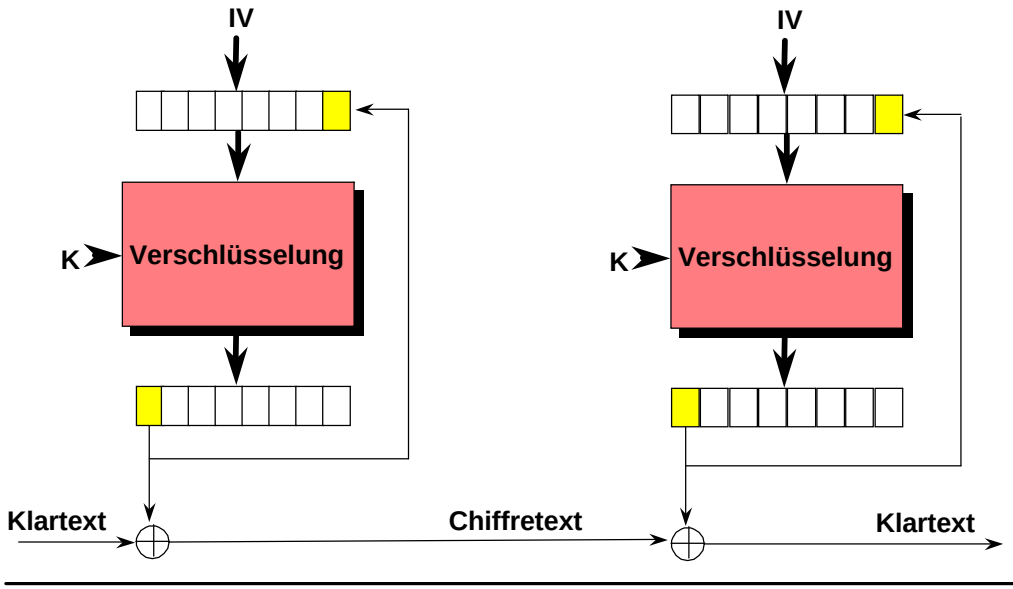
Wenn zudem für jede Nachricht ein zufällig gewählter IV zum Einsatz kommt, können selbst Nachrichten mit identischen Anfangs-stücken nach der Chiffrierung nicht als solche erkannt werden. Der IV braucht zu diesem Zweck nicht geheim gehalten zu werden.

Ein weiterer Vorteil des CBC.Modus besteht darin, dass sich jede Veränderung am Chiffretext beim Entschlüsseln auf die nachfolgenden Blöcke fortpflanzt. Wenn die Nachricht Redundanz enthält, wird dadurch die Wahrscheinlichkeit für das Erkennen von Manipulationen (inklusive Vertauschen, Weglassen und Duplizieren von Blöcken) beträchtlich erhöht.

Falls die übertragenen Nachrichten jedoch keine Redundanz enthalten oder Verfälschungen vom Empfänger aus anderen Gründen nicht leicht erkannt werden können, bietet auch die Verschlüsselung im *Cipher Block Chaining Mode* keinen aus-reichenden Schutz gegen Manipulationen.

K = Key

IV = Initialisation Value

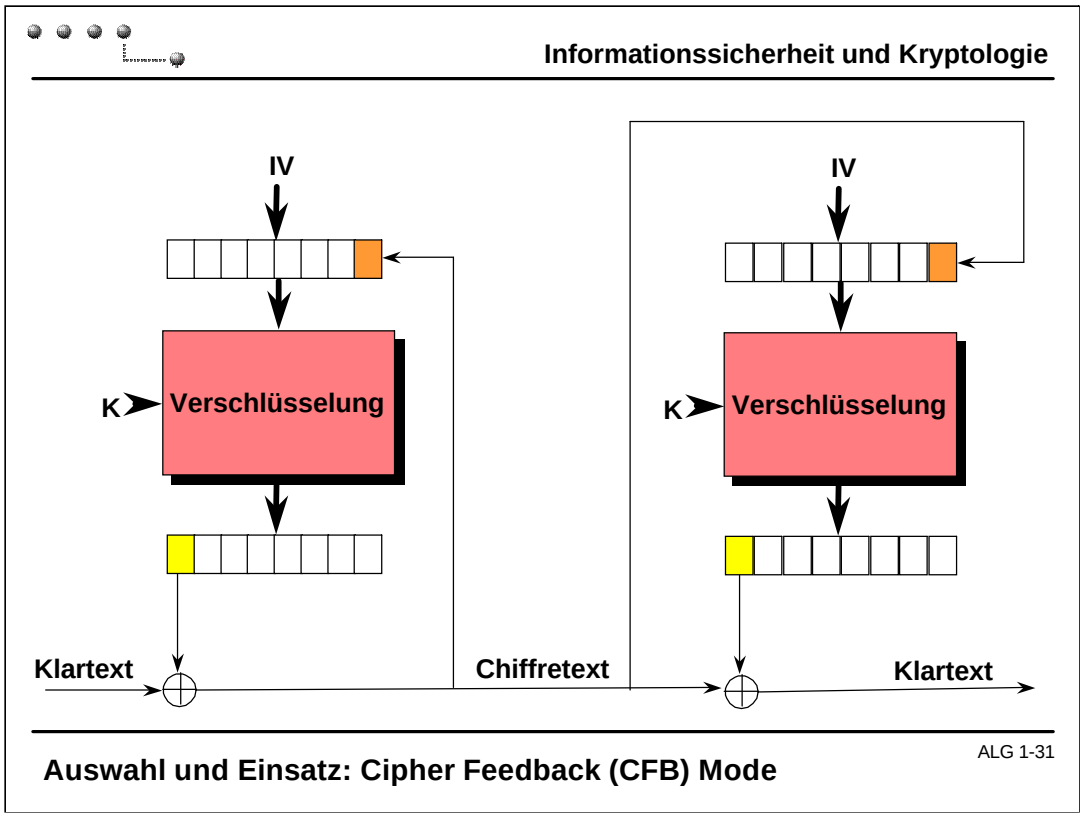


ALG 1-30

Auswahl und Einsatz: Output Feedback (OFB) Mode

Der OFB-Modus ist eine Betriebsart, in der eine Blockchiffre als Pseudozufallsgenerator für eine Stromchiffre dient. Dazu wird jeweils ein m-Bit Block des Outputs der Verschlüsselungsoperation in das mit einem Initialisierungsvektor vorbesetzte Inputregister zurückgekoppelt. Dieser Teilblock stellt auch das jeweils nächste Glied der Schlüsselfolge dar und wird mit m Bit der Klartextfolge XOR-verknüpft.

Der Wert m kann dabei zwischen 1 und der gesamten Blocklänge frei gewählt werden. Der Modus wird dann auch oft mit OFB-m bezeichnet. Zur Verschlüsselung von m Bit des Klartextes ist in dieser Betriebsart eine Verschlüsselungsoperation nötig, so dass der Durchsatz im allgemeinen (und ganz besonders im Fall m=1) nur einen Bruchteil des Wertes für den ECB- oder CBC-Modus erreicht.



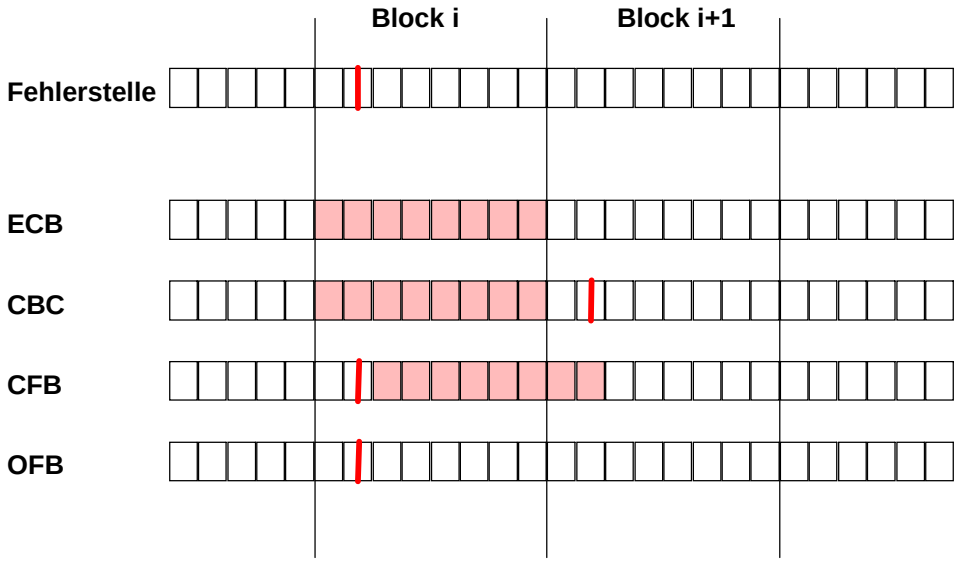
Der CFB Modus ist eine andere Art und Weise, eine Blockchiffre als Stromchiffre einzusetzen. Wie im OFB-m Modus werden auch im CFB-m Modus jeweils m Bit aus dem Ergebnis der Verschlüsselungsoperation mit dem Klartext verknüpft. Zurückgekoppelt wird nun allerdings nicht der Output der Verschlüsselungsoperation, sondern der resultierende Chiffretext.

Durch die Vorwärtskopplung des Chiffretextes beim Empfänger ist der CFB-Modus selbstsynchronisierend, d.h. das System erholt sich von selbst von Synchronisierungsfehlern, sobald die Fehlerstelle aus dem Inputregister des Empfängers heraus-geschoben wurde.

Auf der anderen Seite ergibt sich im CFB-Modus im Gegensatz zum OFB-Modus eine Fehlerfortpflanzung. Ein Fehler wirkt sich ausser auf das betroffene Zeichen auch auf die folgenden so lange aus, bis er das Inputregister verlassen hat.

Die hier vorgestellten Betriebsarten sind wurden als Betriebsarten für DES zusammen mit diesem normiert und können im Detail im entsprechenden NIST-Standard nachgelesen werden, der unter dem später vorgestellten NIST Link bezogen werden kann.

Natürlich kommen die beschriebenen Betriebsarten nicht nur mit DES sondern in derselben Weise auch mit anderen Blockchiffren¨zum Einsatz.



ALG 1-32

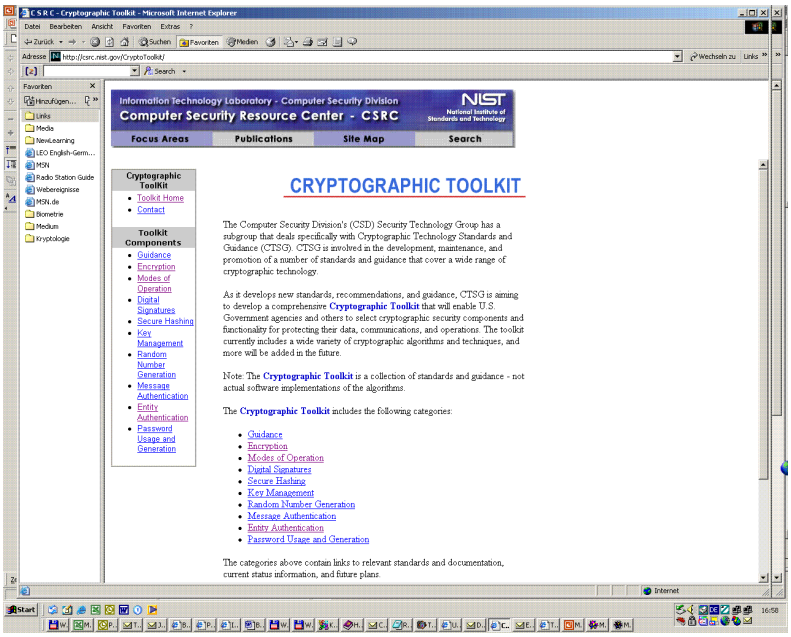
Auswahl und Einsatz: Fehlerfortpflanzung

Die Abbildung fasst das Verhalten bei einem Bitfehler in jeder der Standardbetriebsarten von Blockchiffren zusammen, wobei für m ein Achtel der Blockgrösse gewählt wurde.

Unter Fehlerfortpflanzung versteht man den Effekt, dass ein einzelner Übertragungs-fehler beim Chiffretext durch den Dechiffriervorgang verstärkt wird und sich im Klartext auf einen grösseren Bereich “fortpflanzt”.

Ein wichtiger Vorteil des OFB-Modus ist, dass er im Gegensatz zu allen anderen Betriebsarten keine Fehlerfortpflanzung aufweist. Dies ist zum Beispiel dann von Bedeutung , wenn die Verschlüsselung nachträglich in ein bezüglich seines Fehlerverhaltens optimiertes Kommuni-kationssystem integriert wird.

Blockchiffren sind sehr vielseitig einsetzbar, nicht nur zur Geheimhaltung von Informationen sondern, wie noch gezeigt wird, auch zum Schutz der Integrität von Daten. Für diesen Zweck ist ein gewisses Mass an Fehlerfortpflanzung aber geradezu die Voraussetzung.



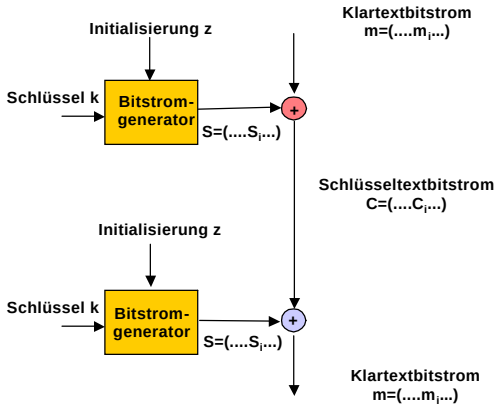
Unter
http://csrc.nist.gov/CryptoToolkit/
Findet man alle 4 offiziell anerkannten Blockchiffren mit detaillierten Spezifikationen:

- DES
- Triple-DES
- Skipjack
- AES

Auswahl und Einsatz: NIST / CSRC

Die Seite mit dem Kryptographic Toolkit enthält sehr viele wichtige Originaldokumente zu den standarisierten Algorithmen und Verfahren für die ICT-Sicherheit.

Stromchiffren



ALG 1-34

Stromchiffren: Schema

Stromchiffren

Zeichen des Klartextes werden bit-weise mit XOR ver- und entschlüsselt

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

$A \oplus A = 0$

$A \oplus 0 = A$

$A \oplus 1 = \overline{A}$

$A \oplus B = C$

$B \oplus C = A$

Ein Angriff mit Klar- und Schlüsseltext erlaubt sofort die Rekonstruktion des kryptographischen Bit-Stromes S.

Klartext m	0 1 0 1
Kryptographischer Bit-Strom S	0 0 1 1
Schlüsseltext C	0 1 1 0
Schlüsseltext C	0 1 1 0
Kryptographischer Bit-Strom S	0 0 1 1
Klartext m	0 1 0 1

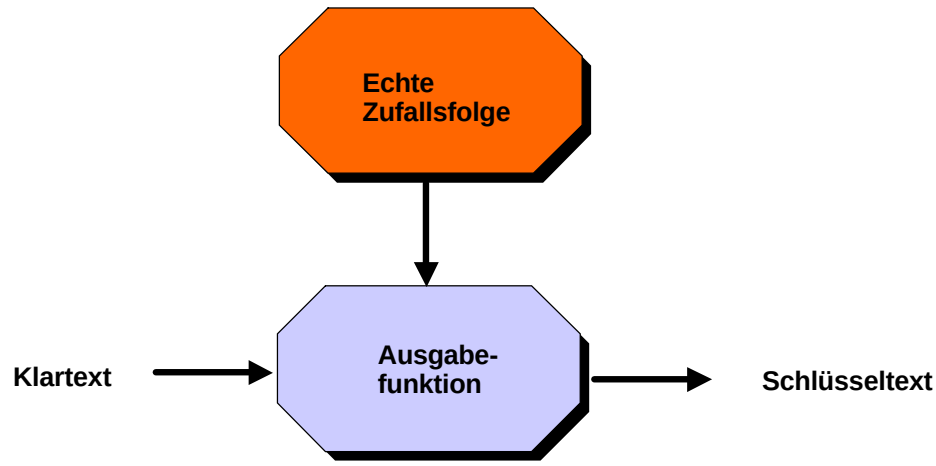
Die kryptographische Sicherheit muss somit in der komplexen Beziehung zwischen S, z und k liegen (gleiche Konfusions und Diffusionskriterien wie für Blockchiffre).

Damit ein Angreifer nicht einfach den Bit-Strom S rekonstruiert, muss S eine sehr lange Periode haben. Dies wird durch die zufällige Initialisierung mit z erreicht.

z ist eine (pseudo) zufällige (nicht unbedingt geheime) Bitfolge

Verkettung

Eine zusätzliche Schwierigkeit ergibt sich durch die Verkettung des Zufallsgenerators z mit der Schlüsseltextausgabe. Damit wird der Schlüsseltext an der Stelle i von allen früheren Stellen abhängig. Die gleiche Operation ist auch für Blockchiffren möglich.



ALG 1-35

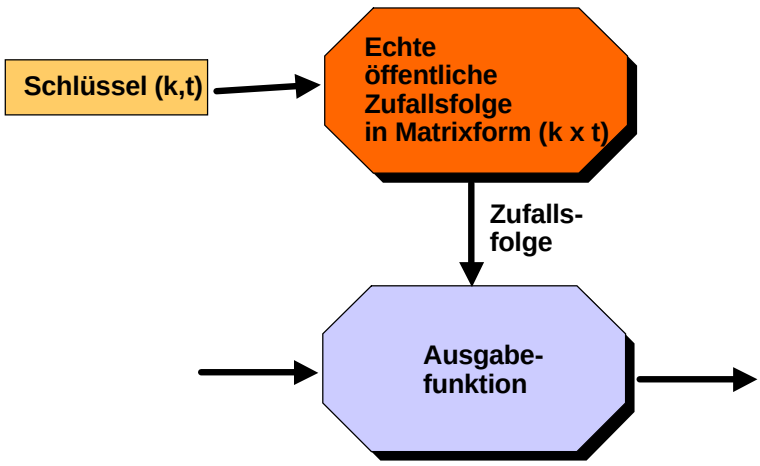
Zufallsfolgen: Pseudozufallsgenerator

Bereits in der Einführung wurde gezeigt, dass ein informationstheoretisch sicheres Kryptosystem realisierbar ist, wenn der Klartext polyalphabetisch verschlüsselt wird und die Schlüsselfolge für die zeichenweise Verschlüsselung eine echte, nur einmal gebrauchte Zufallsfolge ist. Man nennt ein solches Schema:

One-time-pad

Offensichtlicher Nachteil ist die umständliche Schlüsselverwaltung, da jede Nachricht einen Schlüsselstrom von der Länge der Nachricht verbraucht. Ausserdem bedingt ein One-time-pad exakte Synchronisation zwischen Sender und Empfänger.

Ein one-time-pad ist gegenüber jedem kryptoanalytischen Angriff sicher, da jede Nachricht mit der genau gleichen Wahrscheinlichkeit zu einem bestimmen Schlüsseltext führen kann. Wegen der Unmöglichkeit echte Zufallsfolgen beliebiger Länge effizient zu erzeugen ist dieses Verschlüsselungsverfahren nur in Ausnahmefällen brauchbar (z.B. rotes Telephon).



ALG 1-36

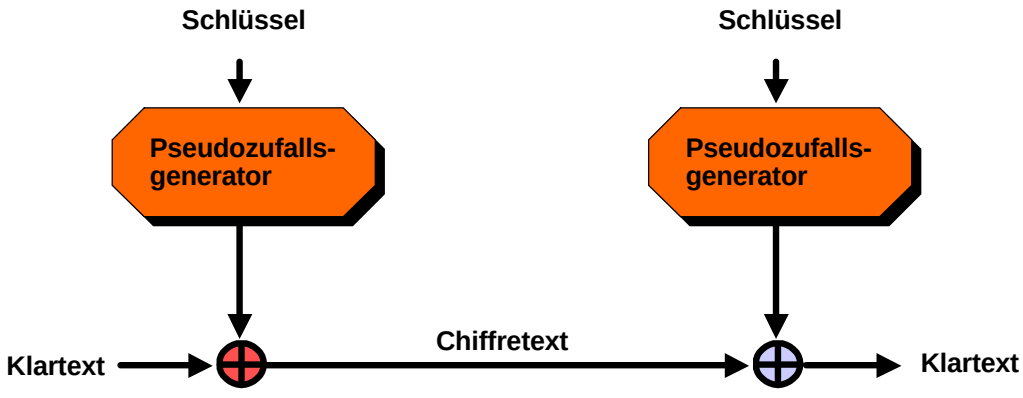
Zufallsfolgen: Pseudozufallsgenerator

Ein auf der Idee des one-time-pad aufbauender polyalphbetischer Substitutionschiffre mit beschränktem Schlüssel geht von einer öffentlich echt zufälligen Bitfolge aus, die in Matrixform mit k Zeilen und t Spalten aufgeschrieben ist (zB. k=50, $t=10^{20}$. Eine echte Zufallsfolge wird damit durch den Schlüssel (k,t) der den Startpunkt angibt definiert.

Ein potentieller Angreifer sieht sich der gleichen Situation wie bei einem echten one-time-pad gegenüber, jedoch verfügt er über die Zusatzinformation der öffentlichen Zufallsfolgenmatrix.

Es bleibt ihm nichts anderes übrig als die möglichen Zufallsfolgenbits als Startpunkte zu inspizieren. Man kann zeigen, dass die Wahrscheinlichkeit, dass der so erzeugte Chiffre sicher ist (falls der Angreifer den Anteil q aller Bits auswertet, n Anzahl Versuche)

$P > 1-nq^k$



ALG 1-37

Zufallsfolgen: Pseudozufallsgenerator

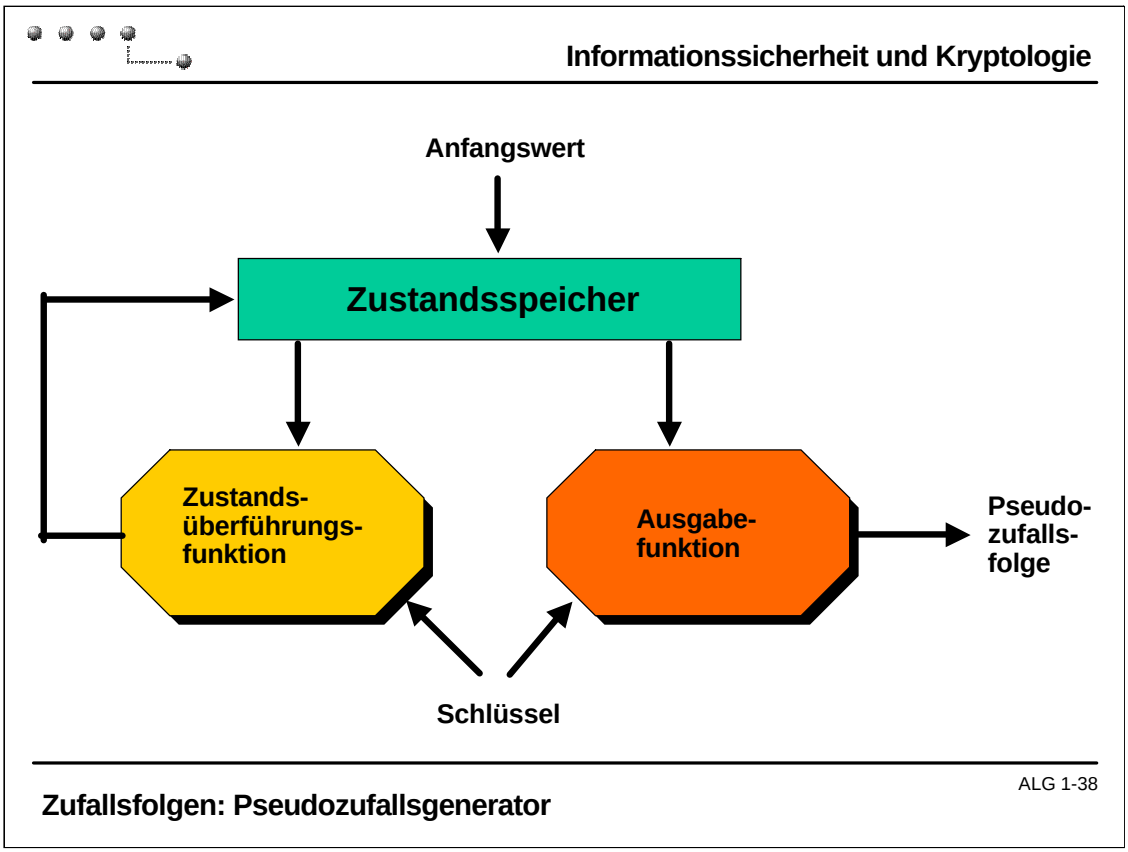
Eine Stromchiffre (engl.: *stream cipher*) oder sequentielle Chiffre ist ein Verschlüsselungsverfahren, bei dem der Klartext zeichenweise mit einer variierenden Funktion verschlüsselt wird.

Wir werden ausschliesslich sogenannte synchrone Stromchiffren betrachten. Bei diesen wird im Gegensatz zum allgemeineren Fall die Pseudozufallsfolge unabhängig vom Klartext erzeugt und dann mit diesem (meist durch Addition mod 2) verknüpft. Durch Addition derselben Folge kann dann beim Empfänger der Klartext aus den Chiffretext wiederhergestellt werden.

Man kann zeigen, dass eine solche Chiffre im informationstheoretischen Sinn absolut sicher wäre, wenn statt einer Pseudozufallsfolge eine tatsächlich zufällig erzeugte Folge aufaddiert würde. Um das Verfahren in der Praxis handhabbar zu machen wird jedoch die Zufallsfolge durch eine auf beiden Seiten der Übertragungs-strecke in Abhängigkeit von einem Schlüssel erzeugte deterministische Folge ersetzt.

Die wichtigsten Vorteile von Stromchiffren sind, dass sie eine geringe Verzögerungszeit beim Ver- und Entschlüsseln erfordern sowie keine Fehlerfortpflanzung aufweisen.

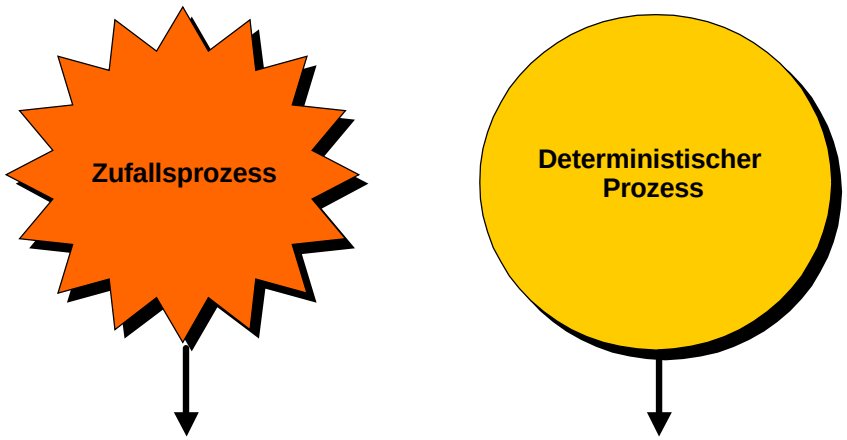
Unter Fehlerfortpflanzung versteht man den Effekt, dass ein Übertragungsfehler beim Chiffretext durch den Dechiffriervorgang verstärkt wird und sich im Klartext auf mehrere Zeichen “fortpflanzt”.



Damit die Verschlüsselungsfunktion einer Stromchiffre sich von Schritt zu Schritt verändern kann, muss sie ausser vom augenblicklichen Klartextzeichen noch von einer anderen Grösse abhängen, die als innerer Zustand bezeichnet wird. Der innere Zustand wird nach jedem Verschlüsselungsschritt durch eine Zustandsüberführungsfunktion verändert. In allen realen Implementierungen kann dabei jeweils nur einer von endlich vielen Zuständen angenommen werden. Die zur Verschlüsselung verwendete Pseudozufallsfolge wird mit Hilfe der Ausgabefunktion aus dem jeweiligen Zustand abgeleitet.

Zur Initialisierung des Pseudozufallsgenerators muss neben dem Schlüssel auch ein Anfangswert für den inneren Zustand (der nach dem englischen *initial value* häufig mit IV bezeichnet wird) eingegeben werden.

Die Sicherheit einer Stromchiffre hängt entscheidend von der Anzahl der möglichen Zustände und der Komplexität der Überführungsfunktion ab.



ALG 1-39

Zufallsfolgen: Pseudozufallsgenerator

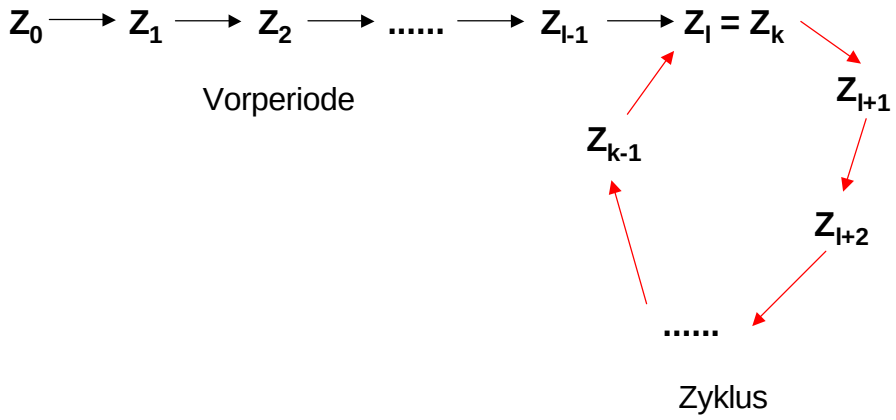
In der Regel nennt man eine Folge von Zeichen zufällig, wenn sie durch ein nichtdeterministisches Verfahren erzeugt wird. Zum Beispiel kann man durch wiederholtes Werfen einer Münze und der Zuordnung Kopf zu 0 und Zahl zu 1 eine binäre Zufallsfolge erzeugen.

Findet das Werfen der Münze unter idealen Bedingungen statt, so sind die Bits der entstehenden Folge stochastisch unabhängig und gleichverteilt. Es ist völlig klar, dass bei einem derartigen Zufallsexperiment jede Folge die gleiche Chance hat, als Ergebnis aufzutreten. Diese Tatsache macht es eigentlich unmöglich, die Eigenschaft “Zufälligkeit” einzelnen Folgen zuzuschreiben.

Dies macht es auch schwierig, den Begriff Pseudozufallsfolge befriedigend zu definieren.

Wenn die Art des Zufallsexperiments und damit die Verteilung der betrachteten Folgen bekannt ist, kann man statistische Aussagen über Grössen gewinnen, die man an den erzeugten Folgen beobachtet. Diese können wiederum dazu dienen, statistische Tests zu konstruieren, die von einer gegebenen Folge prüfen, ob sie mit einer hohen Wahrscheinlichkeit von dem betrachteten Zufallsexperiment erzeugt worden sein könnte. Auf diese Weise lassen sich Kriterien für die Definition von Pseudozufalls-folgen ableiten.

Endliche Maschine



Zufallsfolgen: Pseudozufallsgenerator

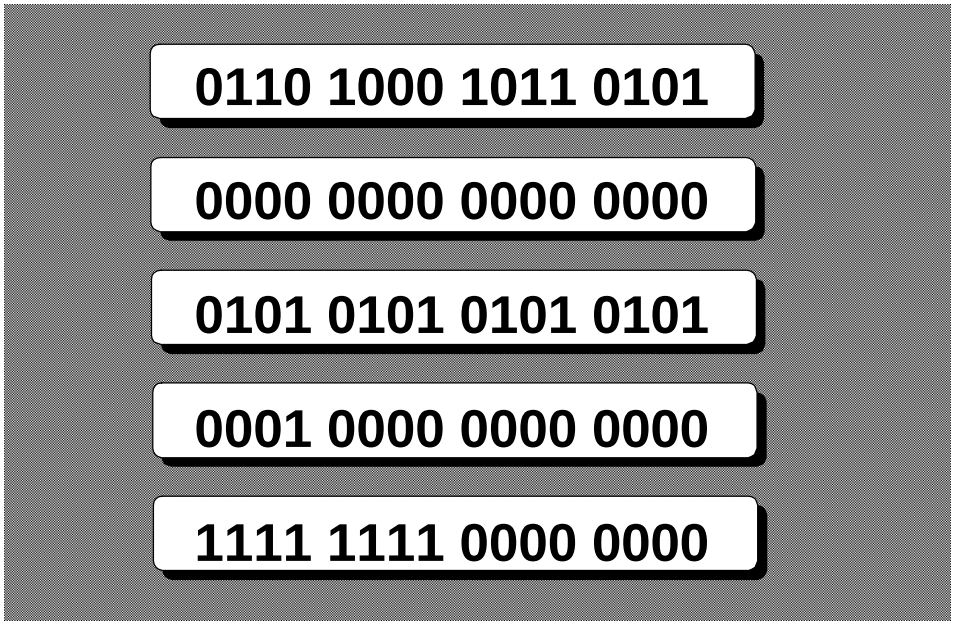
Betrachten wir eine deterministische Maschine, die n verschiedene innere Zustände annehmen kann und autonom, d.h. ohne Eingaben von aussen, arbeitet. Spätestens nach dem n-ten Schritt muss die Maschine in einen Zustand geraten, den sie bereits vorher einmal innehatte. Da das momentane Verhalten der Maschine definitionsgemäss nur vom augenblicklichen Zustand abhängt, folgt dass ab diesem Zeitpunkt immer die gleiche Sequenz von Zuständen durchlaufen wird.

Es bezeichne Z_0 den Anfangszustand, in dem die Maschine gestartet wird, und Z_j den Zustand, in dem sie sich nach dem j-ten Schritt befindet. Sei k der Zeitpunkt der ersten Wiederholung eines Zustands, also $Z_k = Z_l$ (mit $l < k$). Dann heisst der Abschnitt von Zuständen von Z_0 bis Z_{l-1}, die **Vorperiode** der Folge, der Abschnitt von Z_l bis bis Z_{k-1} der **Zyklus** der Folge.

Die Länge des Zyklus heisst die **Periode** der Folge. Eine grosse Periode ist für kryptologische Anwendungen von hoher Bedeutung (siehe Vigenère-Chiffren).

Dabei müssen nicht notwendig alle möglichen Zustände von einem bestimmten Anfangs-zustand aus erreichbar sein.

Der Folgezustand einer endlichen Maschine ist durch die Zustandsüber-führungsfunktion eindeutig bestimmt. Es können mehrere Zustände in denselben münden oder auch mehrere Zustände ohne Vorgänger existieren. Das gesamte Zustandsdiagramm einer endlichen Maschine besteht daher im allgemeinen aus einigen nicht zusammenhängenden Zyklen mit jeweils mehreren Anfangsstücken.



ALG 1-41

Zufallsfolgen: Zufälligkeit endlicher Folgen

Wann ist eine von einer endlichen Maschine erzeugte Folge, die aus Vorperiode und Zyklus besteht, also endlich ist, als “zufällig” oder “pseudozufällig” zu betrachten?

Wenn man endliche Folgen (beispielsweise der Länge k) mittels Münzwurf erzeugt, muss man feststellen, dass jede Folge mit der Wahrscheinlichkeit 2^{-k} als Ergebnis auftreten kann. Dennoch ist man geneigt, manche Folgen, wie z.B. die Folge

0110100010110101

für “zufälliger” zu halten als andere, wie z.B. die Folge

0000000000000000.

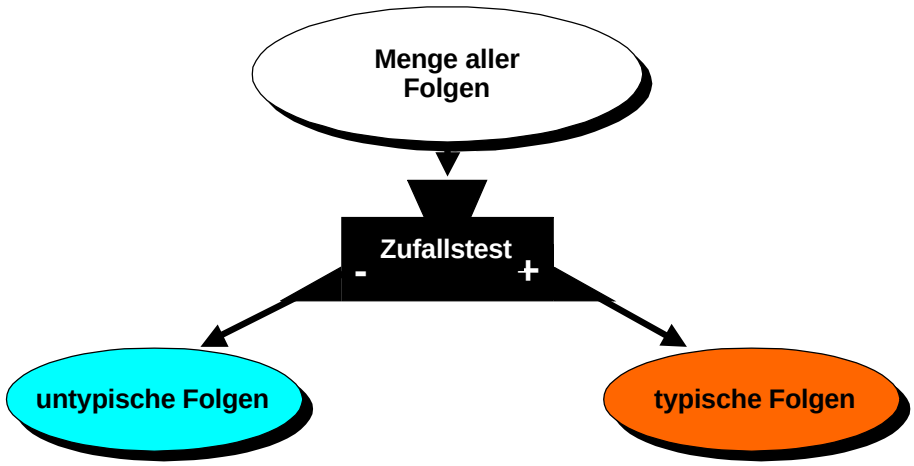
Dies liegt daran, dass die zweite Folge offensichtliche Eigenschaften besitzt, die nur von wenigen binären Folgen der Länge 16 geteilt werden.

Es gibt nur eine einzige Folge der Länge 16, die aus lauter Nullen besteht, aber 12870 verschiedene Folgen, die aus acht Einsen und acht Nullen bestehen. Eine solche Folge kann daher, wenn auch nicht als “zufälliger” so doch als “typischer” bezeichnet werden.

Allerdings kann eine ausgewogene Anzahl von Nullen und Einsen sicher nicht das einzige Kriterium sein, das eine binäre Folge im intuitiven Sinn als zufällig erscheinen lässt, wie das Beispiel

0101010101010101

zeigt.



ALG 1-42

Zufallsfolgen: Zufälligkeit endlicher Folgen

Es wurden verschiedene Kriterien für die Unterscheidung zwischen typischen (und damit eher zufälligen) und untypischen Folgen vorgeschlagen. Die Gefahr besteht dabei, dass durch zu strenge Kriterien die Menge der typischen Folgen leer oder zumindest sehr klein (und damit für kryptologische Zwecke unbrauchbar) wird.

In der Literatur findet man auch statistische Tests, die es zu entscheiden erlauben, ob eine vorliegende Folge von einem Zufallsprozess mit einer gegebenen Verteilung mit hoher Wahrscheinlichkeit erzeugt worden sein könnte. Solche Tests sind nützliche Hilfsmittel für die Beurteilung der Sicherheit von Pseudozufallsgeneratoren und Stromchiffren.

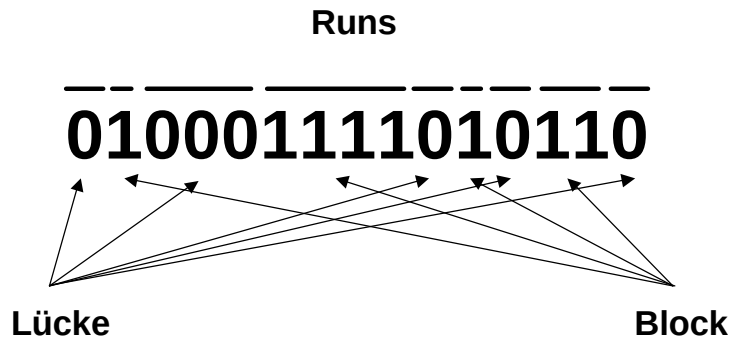
Bezogen auf eine endliche Menge von Folgen (z.B. aller binärer Folgen der Länge k) stellt auch jeder solche Zufallstest eine Aufteilung der Menge in die Teilmenge der (für einen bestimmten Zufallsprozess) typischen Folgen und ihr Komplement, die untypischen Folgen dar.

Golombs Kriterien für Zufälligkeit

G1: **In jedem Zyklus der Folge unterscheidet sich die Anzahl der Einsen von der Anzahl der Nullen um höchstens eins.**

Im folgenden werden drei von S.W. Golomb angegebene Pseudozufallskriterien vorgestellt. Wir betrachten dazu unendliche Folgen, die durch Wiederholung einer endlichen Folge der Länge n entstehen, also aus einem Zyklus ohne Vorperiode bestehen.

Das erste Kriterium leitet sich aus der schon erwähnten Beobachtung her, dass man bei wirklich zufälligen Folgen im Mittel die gleiche Anzahl von Nullen und Einsen beobachtet.



Zufallsfolgen: Zufälligkeit endlicher Folgen

Einen Abschnitt aufeinanderfolgender identischer Zeichen maximaler Länge in einer Folge bezeichnen wir als **Run**. Ein Run von Nullen heisst **Lücke**, ein Run von Einsen heisst **Block**.

Die Folge in der Abbildung enthält 9 Runs, und zwar jeweils 5 Lücken und 4 Blöcke mit folgenden Längen:

4 Lücken der Länge 1
1 Lücke der Länge 3

2 Blöcke der Länge 1
1 Block der Länge 2
1 Block der Länge 4

Golombs Kriterien für Zufälligkeit

G1: **In jedem Zyklus der Folge unterscheidet sich die Anzahl Einsen von der Anzahl der Nullen um höchstens eins.**

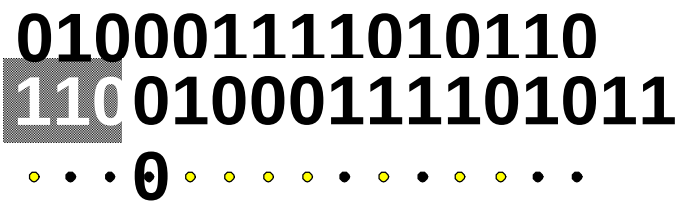
G2: **In einem Zyklus haben $1/2^i$ aller Runs die Länge i. Für jede Länge gibt es gleich viele Blöcke und Lücken.**

Das zweite Kriterium stellt eine Forderung an die Anzahlen der Blöcke und Lücken in einem Zyklus der Folge. In einer echten Zufallsfolge erwartet man, dass etwa die Hälfte der Runs die Länge 1, ein Viertel die Länge 2, ein Achtel die Länge 3 usw. haben. Natürlich treten jeweils etwa gleichviele Blöcke und Lücken der jeweiligen Längen auf.

G2 muss für alle diejenigen i erfüllt sein, für die der gesamte Zyklus mindestens 2^{i+1} Runs enthält.

Übungsaufgabe:

Erfüllt die Folge, deren Zyklus aus der im vorigen Beispiel gegebenen Folge besteht das Kriterium G2?



$$C(3) = \frac{A(3) - D(3)}{n} = \frac{7 - 8}{15} = -\frac{1}{15}$$

Zufallsfolgen: Zufälligkeit endlicher Folgen

Die Autokorrelationsfunktion C(t) einer Folge mit Periode n ist definiert als:

Sei A(t) die Anzahl der übereinstimmenden Glieder pro Zyklus zwischen der betrachteten Folge selbst und der um t Glieder verschobenen Folge.

Sei D(t) die Anzahl der nicht übereinstimmenden Glieder pro Zyklus.

Dann ist C(t) = (A(t)-D(t)) / n

Golombs Kriterien für Zufälligkeit

G1: In jedem Zyklus der Folge unterscheidet sich die Anzahl Einsen von der Anzahl der Nullen um höchstens eins.

G2: In einem Zyklus haben $1/2^i$ aller Runs die Länge i. Für jede Länge gibt es gleich viele Blöcke und Lücken.

G3: Die Autokorrelationsfunktion C(t) ist konstant für alle Werte von t zwischen 1 und n-1.

Für t = 0 ist der Wert der Autokorrelations-funktion für jede Folge offensichtlich gleich 1. Für jeden anderen Wert kann man bei einer zufälligen Folge erwarten, dass die Paare von Werten (0,0), (0,1), (1,0) und (1,1) jeweils mit relativer Häufigkeit 1/4 auftreten. Die Autokorrelationsfunktion einer zufälligen Folge wird also für t > 0 jeweils einen Wert nahe bei Null annehmen.

Dies gibt Anlass für das dritte Golomb'sche Kriterium.

Die drei Kriterien Golombs sind sehr strikt, da sie die Erfüllung genau vorgegebener Werte fordern, während man bei wirklich zufälligen Folgen meist nur die ungefähre Übereinstimmung mit diesen Werten beobachtet. Es fragt sich daher, ob überhaupt und wie viele Folgen die Glomb'schen Kriterien erfüllen.

Ein einfaches Beispiel stellt die schon zur Illustration der Kriterien gebrauchte Folge dar. Darüber hinaus gibt es jedoch sogar sehr grosse Klassen von Folgen, die alle drei Kriterien erfüllen.

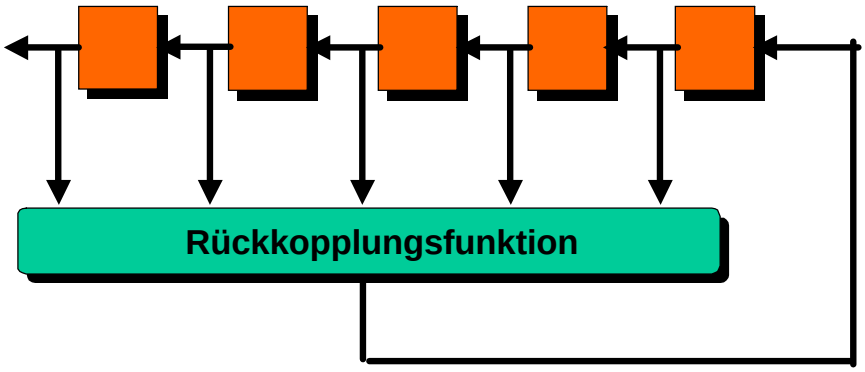
01000111101011O???

Zufallsfolgen: Zufälligkeit endlicher Folgen

Für die Kryptologie ist noch eine andere Eigenschaft von Folgen sehr wichtig, nämlich die der Vorhersagbarkeit der jeweils nächsten Folgenglieder. Werden die Glieder echt zufällig, unabhängig und gleichverteilt erzeugt, so ist eine Vorhersage des jeweils nächsten Zeichens nur durch Raten mit einer Erfolgswahrscheinlichkeit von 0.5 möglich.

Im Fall einer deterministischen Erzeugung hängen die Folgenglieder jedoch von ihren Vorgängern ab. Ein Angriff mit bekanntem Klartext bedeutet im Fall von Stromchiffren, dass der Gegner einen Abschnitt der Pseudozufallsfolge kennt. Für die Beurteilung der Sicherheit einer Stromchiffre kommt es daher entscheidend darauf an, wie schwierig es ist, aus dieser Kenntnis weitere Folgenglieder vorherzusagen.

Eine hohe Periode und gute statistische Eigenschaften sind dabei notwendige, aber - wie wir noch sehen werden - nicht hinreichende Voraussetzungen dafür, dass die Vorhersage schwierig ist.



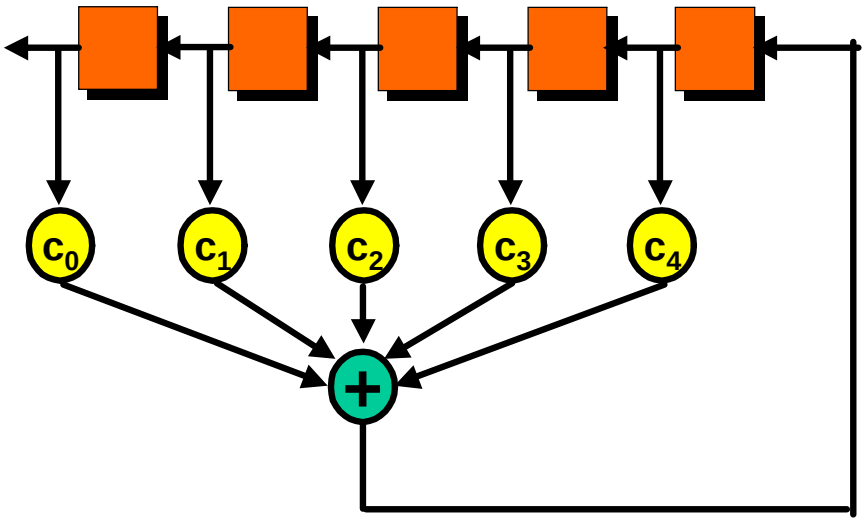
ALG 1-49

Lineare Schieberegister: Rückkoppelung

Für die Realisierung endlicher Automaten durch elektronische Schaltwerke bietet sich die Darstellung ihrer Zustände durch binäre Speicherbausteine (Flip-Flops) an. Verfügt eine Maschine über n solche Speicherzellen, so kann sie 2^n verschiedene Zustände annehmen. Die Zustandsüberführungsfunktion kann in diesem Fall durch n boolesche Schaltfunktionen dargestellt werden, die jeweils die Menge $\{0,1\}^n$ in die Menge {0,1} abbilden.

Eine einfachere Implementierung und Beschreibung erhält man durch die Beschränkung auf rückgekoppelte Schieberegister. Ein Schieberegister besteht aus einer Reihe seriell gekoppelter Speicherzellen. Bei jedem Takt wird der Inhalt der Zellen um eine Stelle weitergeschoben. Der Inhalt der letzten Zelle kann als Ausgabewert betrachtet werden. Die Eingabe für die erste Speicherzelle wird bei einem rückgekoppelten Register durch die sogenannte Rückkopplungsfunktion bestimmt.

Betrachtet man lediglich lineare Rückkopplungsfunktionen, so ergibt sich das folgende Bild.



Lineare Schieberegister: Rückkoppelung

Eine Funktion f von $\{0,1\}^n$ nach $\{0,1\}^m$ heisst linear, wenn gilt:

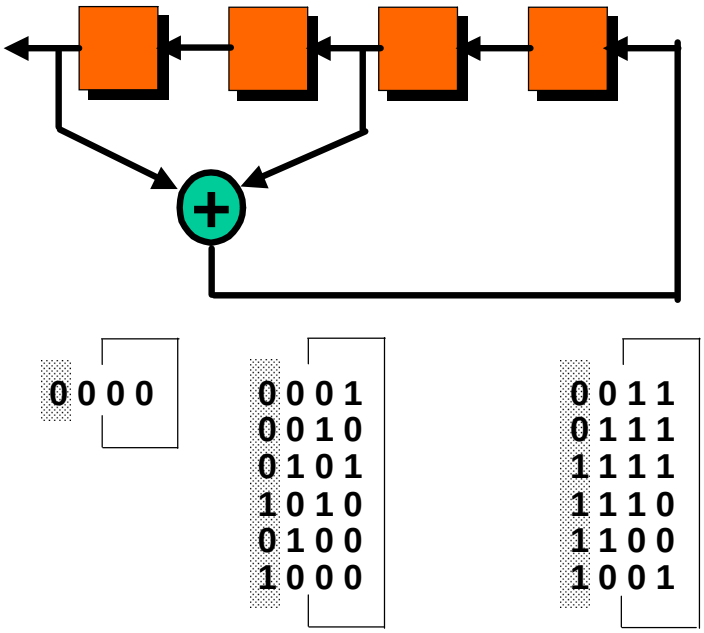
$f(a\underline{x} + b\underline{y}) = af(\underline{x}) + bf(\underline{y})$

für alle a,b aus {0,1} und alle $\underline{x},\underline{y}$ aus $\{0,1\}^n$

Es ist eine wohlbekannte Tatsache, dass eine lineare Abbildung von $\{0,1\}^n$ nach $\{0,1\}^m$ als m x n-Matrix über {0,1} dargestellt werden kann. Für m=1 ergibt sich insbesondere die Darstellung:

$f(x_0,x_1,\ldots,x_{n-1}) = c_0x_0 + c_1x_1 + \ldots + c_{n-1}x_{n-1}$

Im binären Fall nehmen die Koeffizienten lediglich die Werte 0 oder 1 an, so dass sich die Darstellung eines konkreten, linear rückgekoppelten Schieberegisters weiter vereinfacht.



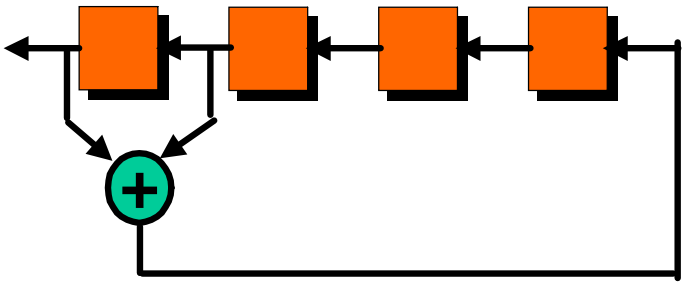
ALG 1-51

Lineare Schieberegister: Rückkoppelung

Die Abbildung zeigt ein Beispiel für ein linear rückgekoppeltes Schieberegister mit der Rückkopplungsfunktion

$$f(x_0,x_1,x_2,x_3) = x_0 + x_2$$

Insgesamt kann ein Schieberegister der Länge n 2^n verschiedene Zustände annehmen. Bei einer linearen Rückkopplung ist klar, dass der Zustand 0 immer in sich selbst abgebildet wird. Die maximale Periode einer mit Hilfe eines linear rückgekoppelten Schieberegisters der Länge n erzeugbaren Folge beträgt demnach 2^n-1.



Folgen mit maximalem Zyklus

0 0 0 0

0 0 0 1
0 0 1 0
0 1 0 0
1 0 0 1
0 0 1 1
0 1 1 0
1 1 0 1
1 0 1 0
0 1 0 1
1 0 1 1
0 1 1 1
1 1 1 1
1 1 1 0
1 1 0 0
1 0 0 0

ALG 1-52

Lineare Schieberegister: Rückkoppelung

Lineare Schieberegisterfolgen mit maximaler Periode 2^n-1 werden als **maximale Folgen** oder kurz **m-Folgen** bezeichnet.

Neben der für kryptologische Anwendungen sehr wichtigen langen Periode (wächst exponentiell mit dem Implementierungsaufwand) verfügen m-Folgen auch über angenehme statistische Eigenschaften, die sie als Pseudozufallsfolgen eine gute Wahl erscheinen lassen. Insbesondere erfüllen alle m-Folgen die drei Golomb'schen Kriterien.

Darüber hinaus gibt es eine genügend grosse Anzahl verschiedener Rückkopplungsfunktionen, die m-Folgen erzeugen, so dass diese als Schlüssel für eine Stromchiffre dienen können.

Alles in allem erscheinen linear rückgekoppelte Schieberegister an diesem Punkt als eine gute Wahl für die Anwendung als Pseudozufallsgeneratoren in Stromchiffren.

Berechnung von linearen Schieberegistern

$$\begin{pmatrix} s_k & s_{k+1} & \cdots & s_{k+n-1} \\ s_{k+1} & s_{k+2} & \cdots & s_{k+n} \\ \vdots & \vdots & & \vdots \\ s_{k+n-1} & s_{k+n} & \cdots & s_{k+2n-2} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} s_{k+n} \\ s_{k+n+1} \\ \vdots \\ s_{k+2n-1} \end{pmatrix}$$

Darstellung von linearen Schieberegistern: Formale Potenzreihe

$$s(x) := s_0 x^0 + s_1 x^1 + s_2 x^2 + s_3 x^3 + s_4 x^4 + \ldots$$

ALG 1-53

Lineare Schieberegister: Rückkoppelung

Das Problem linearer Schieberegisterfolgen, das sie für kryptologische Zwecke in dieser Form leider doch unbrauchbar macht, ist ihre leichte Vorhersagbarkeit.

Man kann sich leicht überlegen, dass bereits die Kenntnis von 2n Folgengliedern genügt, um eine lineares Gleichungssystem aufzustellen, das die eindeutige Bestimmung der Rück-kopplungskoeffizienten des Registers erlaubt. Da es sich dabei um ein Gleichungssystem spezieller Form handelt, ist seine Lösung mit Hilfe des Berlekamp-Massey Algorithmus sogar besonders effizient (O(nlogn)) möglich.

Da jede lineare Maschine (d.h. eine, die im binären Fall allein durch Flip-Flops und XORs realisiert werden kann) durch ein äquivalentes, rückgekoppltes Schieberegister simuliert werden kann, folgt, dass ein kryptologisch besserer Pseudozufallsgenerator auf jeden Fall nichtlineare Operationen beinhalten muss.

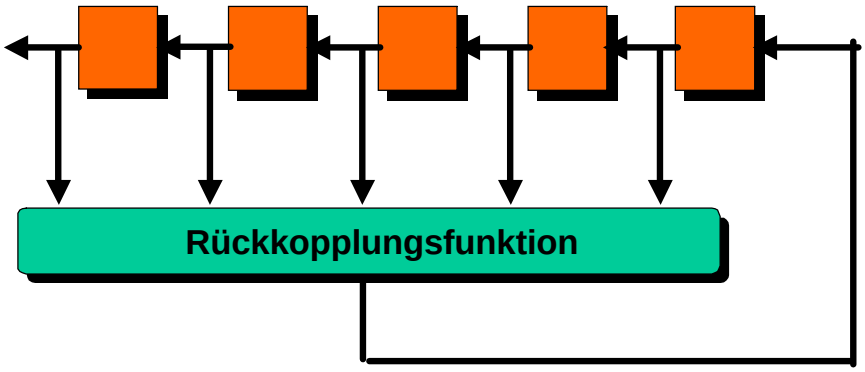
LRSR lassen sich durch formale Potenzreihen darstellen (erzeugende Funktion), deren Grad der Periode des LRSR entspricht. Diese Abbildung erlaubt die Analyse der LRSR mit Hilfe der algebraischen Eigenschaften der Rückkoppelungspolynome.

Übungsaufgabe:

Finden Sie das kürzeste linear rückgekoppelte Schieberegister, das die Folge

1 1 0 0 1 1

erzeugt.



ALG 1-54

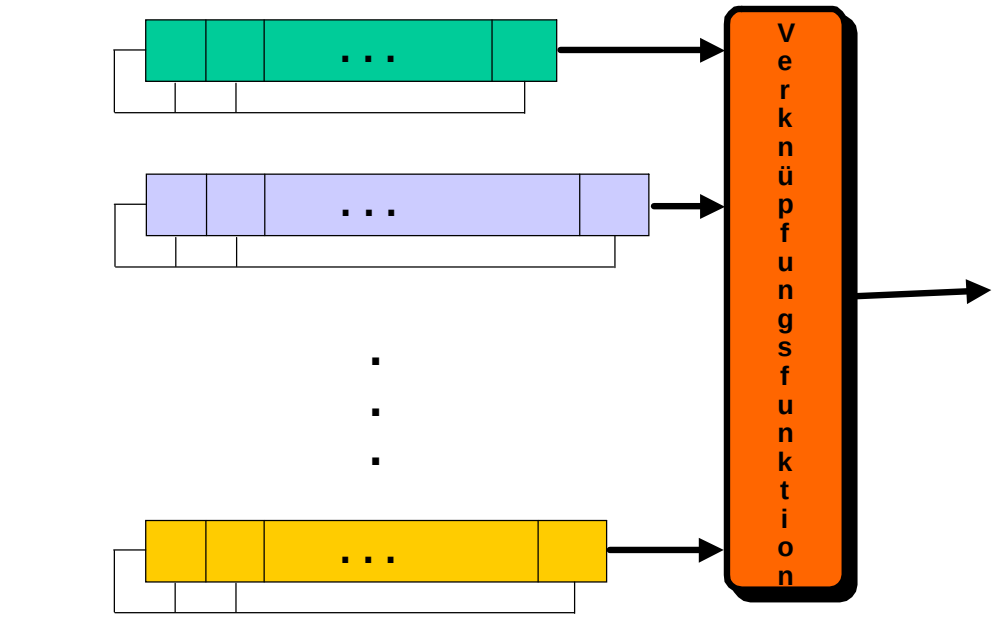
Schieberegister: Lineare Komplexität

Im vorigen Abschnitt wurde festgestellt, dass lineare Schaltungen nicht genügend Sicherheit bieten, um sie zur Erzeugung von Schlüsselfolgen für Stromchiffren heranzuzuiehen. Auch nichtlineare Schaltungen können im Prinzip einfach realisiert werden, insbesondere zum Beispiel auch Schieberegister mit nichtlinearer Rückkopplungsfunktion.

Das Problem dabei ist, dass die Eigenschaften der durch irgendwelche nichtlineare Schaltungen erzeugten Folgen im Gegensatz zum linearen Fall im allgemeinen nicht einfach zu analysieren sind. Um aber verlässliche Aussagen über die kryptologische Stärke eines Verfahrens gewinnen zu können, ist ein gewisses Mass an Analysierbarkeit unverzichtbar.

Hier befindet sich jeder Entwickler eines Kryptosystems in einem gewissen Dilemma. Beschränkt er sich auf eine Klasse gut analysierbarer Verfahren, so scheint dies - wie im linearen Fall - auch den Gegnern die Arbeit zu erleichtern. Versucht er dagegen ein möglichst kompliziertes Verfahren einzusetzen, so bleibt ihm mangels hinreichender Analysemöglichkeit nur die Hoffnung, dass der Algorithmus tatsächlich so komplex ist, wie er erscheint, und es nicht einem Gegner gelingt, eine übersehene Schwäche aufzufinden und auszunutzen.

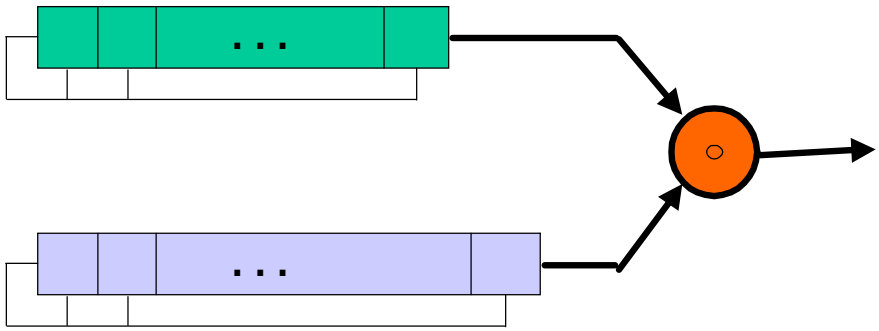
Ein möglicher Weg aus dem beschriebenen Dilemma besteht darin, spezielle Klassen nicht linearer Algorithmen zu betrachten, die eine Analyse ihrer wichtigsten Eigenschaften zulassen.



Schieberegister: Lineare Komplexität

Es liegt nahe, von den gut zu beherrschenden und mit guten statistischen Eigenschaften ausgerüsteten linearen Schieberegisterfolgen (insbesondere m-Folgen) auszugehen und diese in geeigneter Weise durch nichtlineare Funktionen miteinander zu verknüpfen.

Produkt zweier Schieberegisterfolgen



Schieberegister: Lineare Komplexität

Da die Verknüpfung von linearen Schieberegisterfolgen durch Addition (XOR) eine lineare Operation ist, kann man durch diese Operation keinen Zugewinn an Sicherheit erwarten. In der Tat kann man leicht zeigen, dass die so erzeugten Folgen auch mit einem einzigen linear rückgekoppelten Schieberegister erzeugbar sind, dessen Länge lediglich der Summe der beiden verknüpften Schieberegister entspricht.

Das einfachste Beispiel einer nichtlinearen Verknüpfung zweier binären Folgen ist die Multiplikation (modulo 2), die schaltungs-technisch der UND-Funktion entspricht. Dieser in der Abbildung dargestellte Generator wird sich allerdings ebenfalls nicht für die Praxis eignen, da an seinem Ausgang nur in einem Viertel aller Fälle eine Eins, sonst immer eine Null zu erwarten ist.

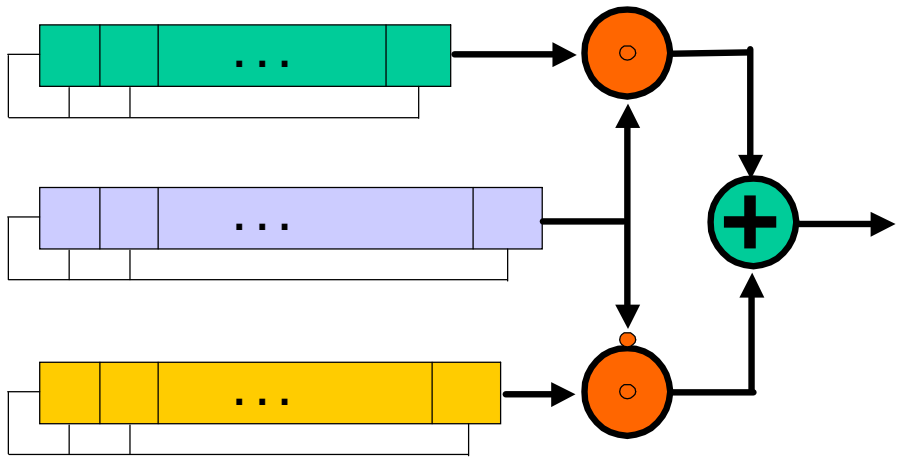
Dennoch ist es wichtig, die Eigenschaften der multiplikativen Verknüpfung zu untersuchen, da sie als Baustein für zusammengesetzte Verknüpfungsfunktionen eingesetzt werden kann.

Bezeichnet man mit L_1 die Länge des Schieberegisters, das die m-Folge S_1 erzeugt und mit L_2 die Länge des Registers zur m-Folge S_2 so gilt, falls $L_1 \# L_2$:

$L(S_1 * S_2) = L_1 * L_2$

$L(S_1 * \neg S_2) = L_1 * (L_2+1)$

Geffe-Generator



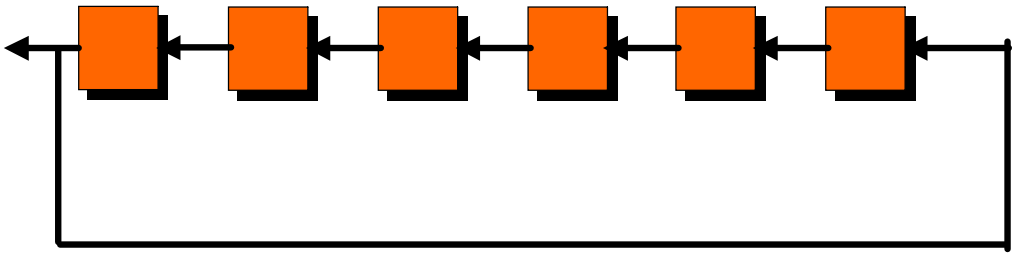
ALG 1-57

Schieberegister: Lineare Komplexität

Ein Beispiel für eine einfache nichtlineare Verknüpfung ist der Generator von Geffe. Seine Arbeitsweise lässt sich anschaulich so beschreiben, dass das mittlere Schieberegister über die beiden UND-Gatter jeweils eines der beiden anderen Schieberegister auf den Ausgang durchschaltet. Damit wird erreicht, dass Nullen und Einsen am Ausgang mit gleicher Häufigkeit auftreten.

Neben einer guten statistischen Verteilung ist natürlich auch die Periode der Ausgangsfolge des Generators von grosser praktischer Bedeutung.

Darüber hinaus stellt sich natürlich die Frage nach weiteren Kriterien für die kryptologische Brauchbarkeit eines Pseudozufallsgenerators.



ALG 1-58

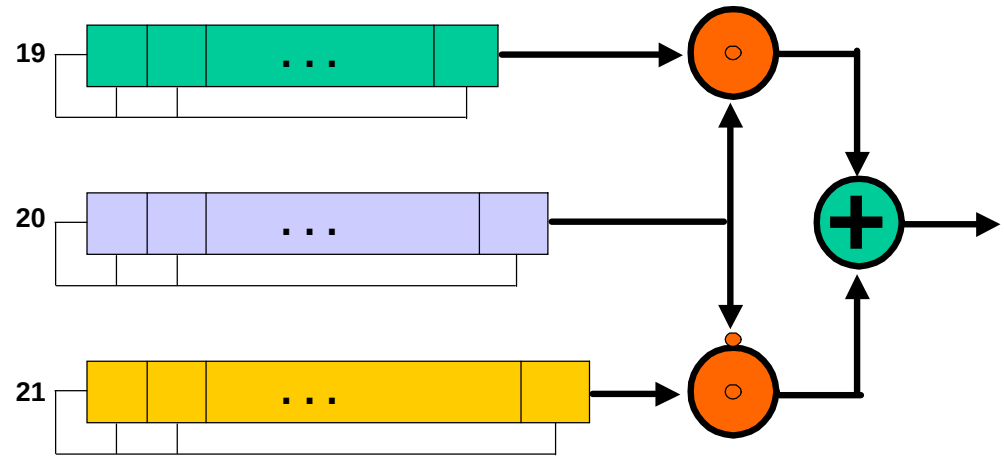
Schieberegister: Lineare Komplexität

Kriterien für die kryptologische Stärke ergeben sich meist aus der Betrachtung möglicher Angriffe und deren Komplexität bei Anwendung auf ein bestimmtes System.

Weiter oben haben wir einen effizienten Angriff auf lineare Pseudozufallsgeneratoren kennengelernt. Mit Hilfe des Berlekamp-Massey Algorithmus gelingt es aus 2n Folgenglieder alle Rückkopplungskoeffizienten zu rekonstruieren. Selbstverständlich ist dieser Angriff nicht auf lineare Generatoren beschränkt, da sich jede periodische Folge - unabhängig von der Art und Weise ihrer Erzeugung - als Ausgabe eines linearen Schieberegisters auffassen lässt. Im Extremfall muss dieses Schieberegister so lang sein, wie die Periode der Folge. In manchen Fällen genügt aber auch ein kürzeres Register.

Die Länge des kürzesten linear rückgekoppelten Schieberegisters, das eine gegebene Folge erzeugen kann, wird als die lineare Komplexität der Folge bezeichnet. Die **lineare Komplexität** einer beliebigen Folge kann samt den zugehörigen Rückkopplungskoeffizienten mit Hilfe des Berlekamp-Massey Algorithmus effizient bestimmt werden.

Eine möglichst hohe lineare Komplexität der erzeugten Folgen ist ein weiteres wichtiges Kriterium für kryptologisch gute Pseudozufallsgeneratoren.



$$f(x_0, x_1, x_2) = x_0x_1 + (x_1+1)x_2 = x_0x_1 + x_1x_2 + x_2$$

$$L(19, 20, 21) = 19 \bullet 20 + 20 \bullet 21 + 21 = 821$$

ALG 1-59

Schieberegister: Lineare Komplexität

Wie schon weiter oben erwähnt ist die lineare Komplexität der Summe zweier Folgen die Summe der linearen Komplexitäten der Folgen.

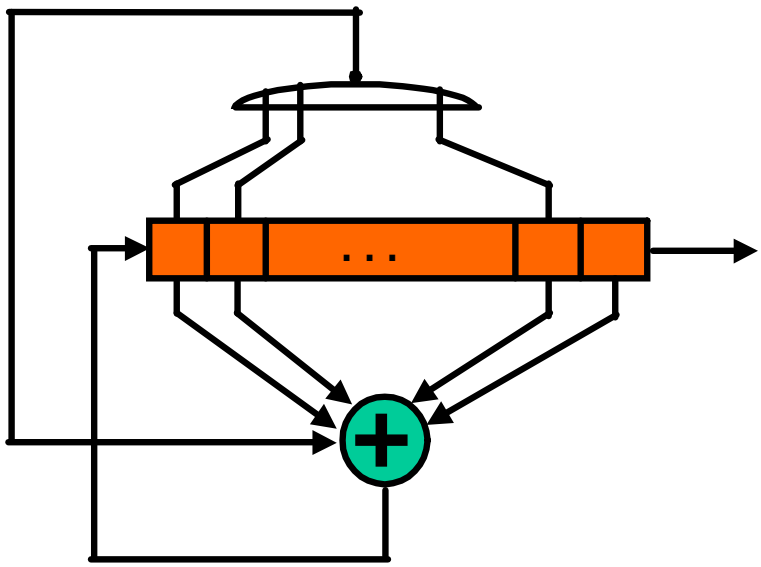
Man kann ebenso zeigen, dass unter bestimmten Voraussetzungen (z.B. Verknüpfung von m-Folgen aus Schieberegistern paarweise verschiedener Längen) die lineare Komplexität des Produktes (UND-Verknüpfung) zweier Folgen das Produkt der linearen Komplexitäten der Folgen ist.

Diese Ergebnisse lassen sich verallgemeinern auf beliebige Verknüpfungsfunktionen, die stets durch Addition und Multiplikation dargestellt werden können.

Im Beispiel eines Geffe-Generators, der aus linear rückgekoppelten Schieberegistern der Längen 19, 20 und 21 zusammengesetzt ist, ergibt sich als die lineare Komplexität der erzeugten Folge der Wert 821. Das bedeutet, das man ein Schieberegister mit 821 Zellen benötigen würde, um dieselbe Folge nur mit linearen Operationen zu erzeugen (im Gegensatz zu 19 + 20 + 21 = 60 Zellen im nichtlinearen Fall.

Es bedeutet weiterhin, dass die gesamte Folge aus 2 mal 821 = 1642 Gliedern vorhersagbar ist.

Durch geeignete Auswahl der Verknüpfungsfunktion und der Längen der verknüpften Register gelingt es mit Hilfe dieser Technik mit vertretbarem Aufwand Pseudozufallsfolgen zu erzeugen, die gegen einen Angriff mit Berlekamp-Massey hinreichend sicher sind.



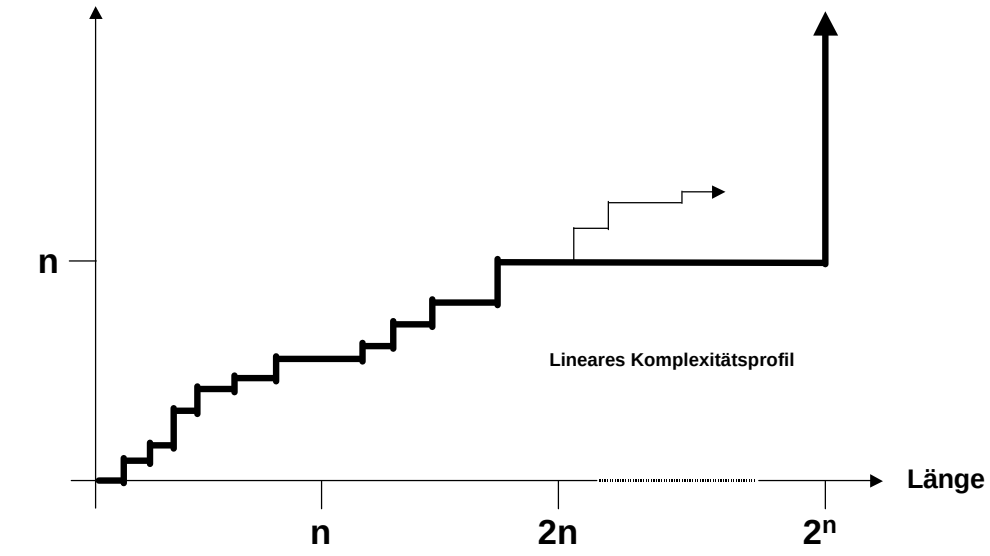
ALG 1-60

Schieberegister: Lineare Komplexität

Die lineare Komplexität darf als Mass für die Sicherheit von Pseudozufallsgeneratoren nicht überbewertet werden. Sie misst lediglich den Aufwand für ein bestimmtes Kryptoanalyseverfahren und stellt damit wie schon die Periode nur ein notwendiges aber keinesfalls hinreichendes Kriterium für die Sicherheit des Generators dar.

Ein Beispiel für einen Generator, der eine grosse Periode, gute statistische Eigenschaften und eine hohe lineare Komplexität in sich vereint, aber nicht sicher ist, zeigt die obige Abbildung.

Die erzeugte Folge ist eine lineare Schieberegisterfolge, in die lediglich eine einzige Null zusätzlich eingefügt wurde. Man kann zeigen dass diese Folge der Periode 2^n (mit n als der Länge des Schieberegisters) auch die lineare Komplexität 2^n besitzt. Dennoch kann die Folge mit den gleichen Methoden wie lineare Folgen aus 2n Folgengliedern fast ganz vorhergesagt werden.



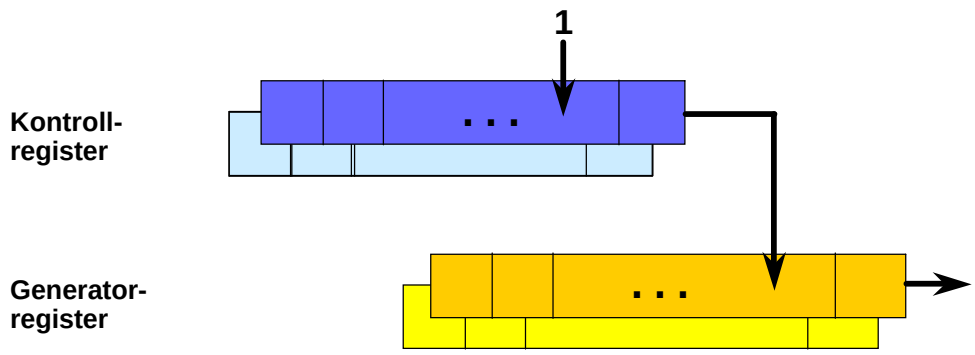
ALG 1-61

Schieberegister: Lineare Komplexität

Statt nur die lineare Komplexität der gesamten Folge zu betrachten ist es aufschlussreicher, die lineare Komplexität für jedes Anfangsstück zu berechnen. Den Graphen dieser Abbildung bezeichnet man als das **lineare Komplexitätsprofil** der Folge.

Wenn man die im vorigen Beispiel betrachtete Folge so anordnet, dass der Run von n Nullen (also die einzig Abweichung von der Linearität ganz am Ende zu stehen kommt, ergibt sich offensichtlich das obige, doch sehr spezielle Bild für das Komplexitätsprofil der Folge.

Man kann zeigen, dass das lineare Komplexitätsprofil zufällig erzeugter Folgen eine völlig andere Gestalt hat, nämlich sich mit relativ geringer Varianz um den Mittelwert n/2 bewegt.



Stromchiffren: Taktgesteuerte Schieberegister ALG 1-62

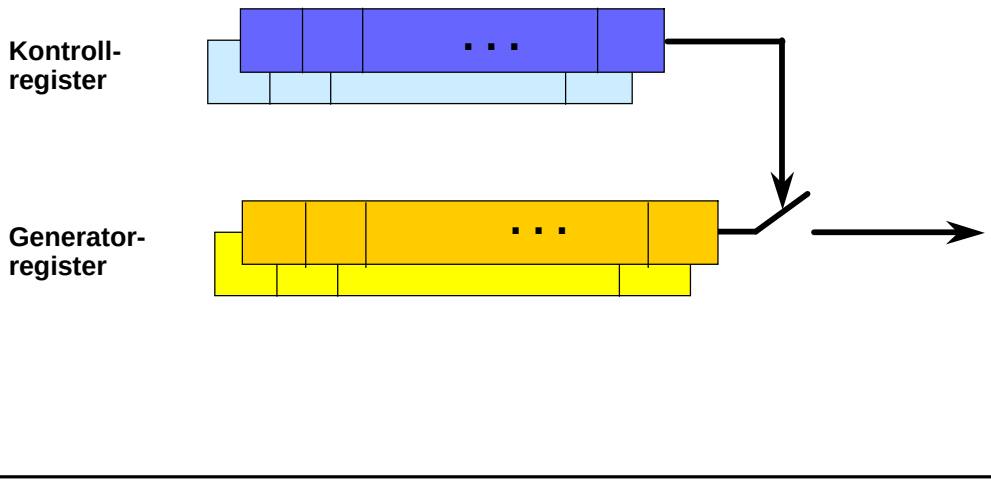
Beim Geffe-Generator und anderen ähnlichen Beispielen wird stets vorausgesetzt, dass alle beteiligten Schieberegister mit jedem Takt einen neuen Zustand annehmen. Auf die naheliegendste Weise kann man dies verallgemeinern, indem man einen Taktsteuereingang vorsieht, der dafür sorgt, dass ein Schieberegister beim nächsten Takt nur dann weitergeschoben wird, wenn an diesem Eingang eine Eins anliegt. Der Taktsteuereingang liefert eine weitere Möglichkeit zur Verknüpfung (linearer) Schieberegister.

Steuert man den Taktsteuereingang eines Schieberegisters mit der Ausgangsfolge eines zweiten, erhält man den sogenannten **Stop-and-Go-Generator**, der allerdings in dieser Form für den Einsatz in Stromchiffren ungeeignet ist.

Zum einen weisen die erzeugten Folgen ungünstige statistische Eigenschaften auf (zu häufige Wiederholungen), zum anderen kann ein Gegner aus jedem Bitwechsel in der Ausgangsfolge auf eine Eins in der Kontrollfolge schliessen und daraus die Parameter des Kontrollregisters erschliessen.

Die Probleme des Stop-and-Go-Generators können vermieden werden, wenn man statt der einfachen Taktsteuerung beim sogenannten **Step-Once-Twice-Generator** das Generatorregister abhängig vom Kontrollregister je einmal oder zweimal taktet (d.h. im letzteren Fall ein Folgenglied überspringt).

Über den Step-Once-Twice-Generator sind keine negativen kryptologischen Eigenschaften bekannt. Als Nachteil gilt jedoch, dass die effektive zur Verschlüsselung zur Verfügung stehende Taktrate durch die Doppelschritte halbiert wird.



Stromchiffren: Shrinking Generator

ALG 1-63

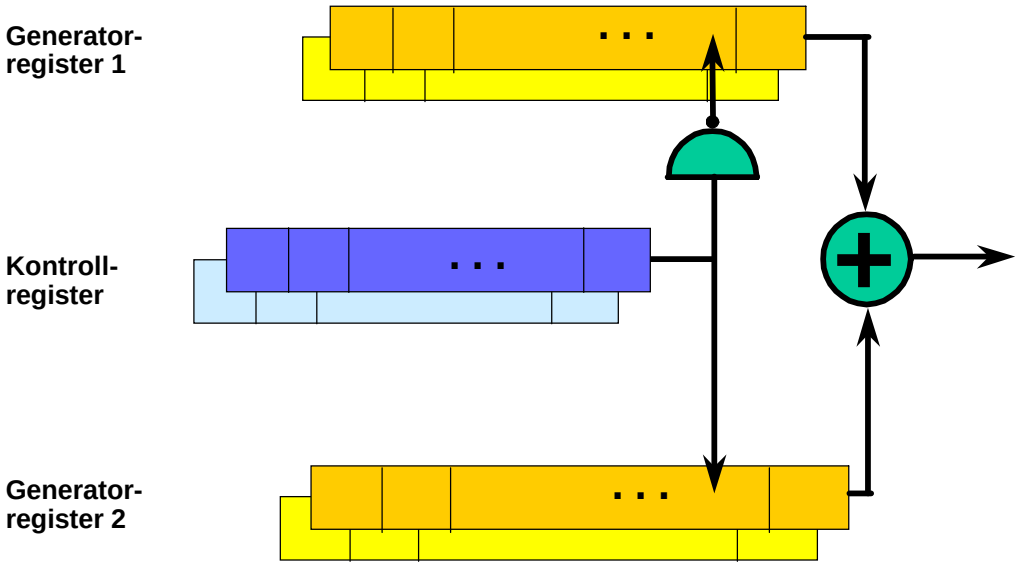
Der Vergleich der beiden vorangehenden Beispiele zeigt, dass kryptologische Stärke eher durch Auslassen als durch Wiederholen von Folgengliedern erzielt werden kann.

Diese Erkenntnis liegt auch einem neueren Vorschlag zugrunde, Beim Shrinking Generator wird ebenfalls die Folge des Generatorregisters in Abhängigkeit vom Kontrollregister verkürzt. Dies jedoch auf wesentlich variablere Weise als beim Step-Once-Twice-Generator, nämlich nach der folgenden Regel:

Ist das i-te Bit der Kontrollfolge eine Eins, so wird das i-te Bit der Generatorfolge in die Ausgangsfolge übernommen, im anderen Fall wird es übersprungen.

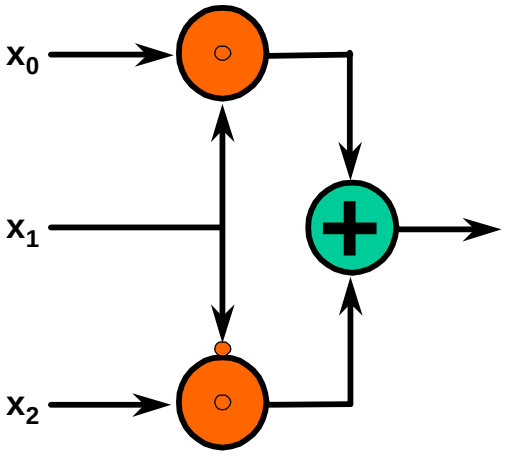
Handfeste kryptologische Vorteile des Schrinking-Generators gegenüber dem Step-Once-Twice-Generator sind (noch) nicht bekannt. Intuitiv scheint jedoch die weniger lokale Anwendung des “Verkürzungsprinzips” ein Fortschritt zu sein.

Ein praktischer Nachteil ist dagegen die weniger gleichmässige Rate der Biterzeugung, die einen gewissen Schaltungstechnischen Aufwand für die Pufferung erfordert.



Stromchiffren: Alternating Stop-and-Go Generator ALG 1-64

Der Alternating Stop-and-Go Generator, auch Wechselschritt-Generator genannt, ist eine andere Weiterentwicklung der Stop-and-Go Idee, bei der es gelungen ist, die kryptologischen Eigenschaften zu verbessern ohne gleichzeitig den Durchsatz zu reduzieren.



x_0	x_1	x_2	$f(x_0,x_1,x_2)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Stromchiffren: Korrelationsattacken

Eine wichtige Klasse von Angriffen auf Pseudozufallsgeneratoren, die aus linearen Schieberegistern zusammengesetzt sind, stellen die Korrelationsattacken dar.Das Prinzip der Korrelationsattacken wird am Beispiel des Geffe-Generators dargestellt.

Man erkennt an der Wertetabelle der Verknüpfungsfunktion, dass der Funktionswert in sechs der acht Zeilen mit dem Wert der x_0-Spalte übereinstimmt. Man kann daher erwarten, dass die Pseudzufallsfolge am Ausgang in 3/4 aller Fälle mit der vom obersten Schieberegister erzeugten Folge übereinstimmt. Beim Vergleich zweier unabhängiger Zufallsfolgen wäre dagegen nur in der Hälfte aller Fälle eine Übereinstimmung zu erwarten. Dieser Effekt lässt sich für die Kryptoanalyse des Geffe-Generators ausnutzen.

Man berechnet dazu die Korrelationen sämtlicher möglicher Schieberegisterfolgen einer gewissen Länge mit einem Stück der Ausgangsfolge. Hat man die falschen Rückkopplungskoeffizienten oder den falschen Anfangszustand gewählt, so wird die damit erzeugte Folge mit hoher Wahrscheinlichkeit keine signifikante Übereinstimmung aufweisen.

Wenn man jedoch die tatsächlich verwendeten Parameter ausprobiert, so wird man die erwarteten 75% Übereinstimmung mit der Ausgangsfolge erhalten. Je länger das für den Vergleich zur Verfügung stehende Folgenstück ist, mit desto grösserer Zuverlässigkeit kann man auf diese Weise die korrekte Eingangsfolge ermitteln. Nach demselben Prinzip kann man anschliessend die Parameter des Schieberegisters am x_2-Eingang ermitteln.

Die Anzahl der zu testenden Fälle ist dabei zwar immer noch sehr hoch, wird jedoch immerhin vom Produkt der Anzahl möglicher Schieberegister auf deren Summe reduziert.

Korrelationsattacken sind auch auf taktgesteuerte Schieberegister möglich. So ist zum Beispiel auch beim Alternating Stop-and-Go Generator die Ausgangsfolge mit der Kontrollfolge korreliert. Im praktischen Einsatz muss dies bei der Wahl der Parameter berücksichtigt werden.

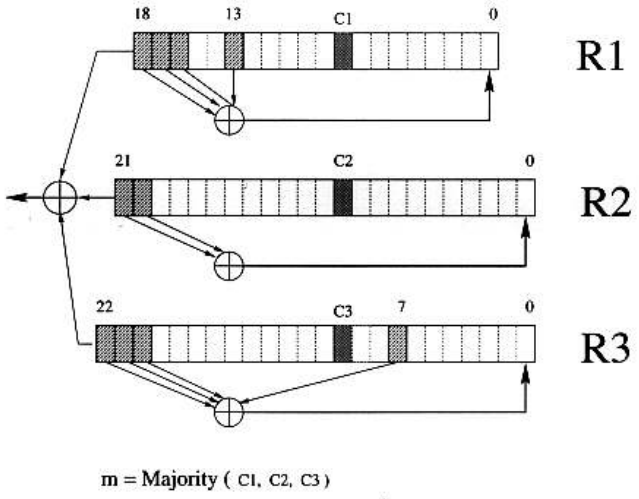


Figure 1: The A5/1 stream cipher.

Stromchiffren: Weitere Beispiele A5-1

Die Abbildung zeigt ein praktisches Beispiel für einen Pseudozufallsgenerator, der im wesentlichen aus linear rückgekoppelten Schieberegistern aufgebaut ist. Der Generator entspricht dem geheimen, im GSM eingesetzten Verschlüsselungsalgorithmus A5-1 (stärkere Variante).

Er besteht aus drei m-Schieberegistern der Länge 19, 22 und 23 mit Rückkoppelungskoeffizienten an den markierten Stellen.

Die Schieberegister operieren im **stop-and-go** Modus gesteuert durch die drei Taktbits (C1, C2, C3). Dabei werden die 2 oder 3 Register getacktet, deren Clock-Bit (Cx) gleich dem Mehrheitswert der drei Clockbits ist (jedes Register im Mittel in 3 von 4 Zyklen).

Der Output des Generators ergibt sich durch die XOR-Funktion der drei msb der drei Register (bit 18, 21 und 22).

Der Generator wird durch den Schlüssel K (max. Länge von 64 bits) und den Framecounter Fn im GSM (22 bits) initialisiert.

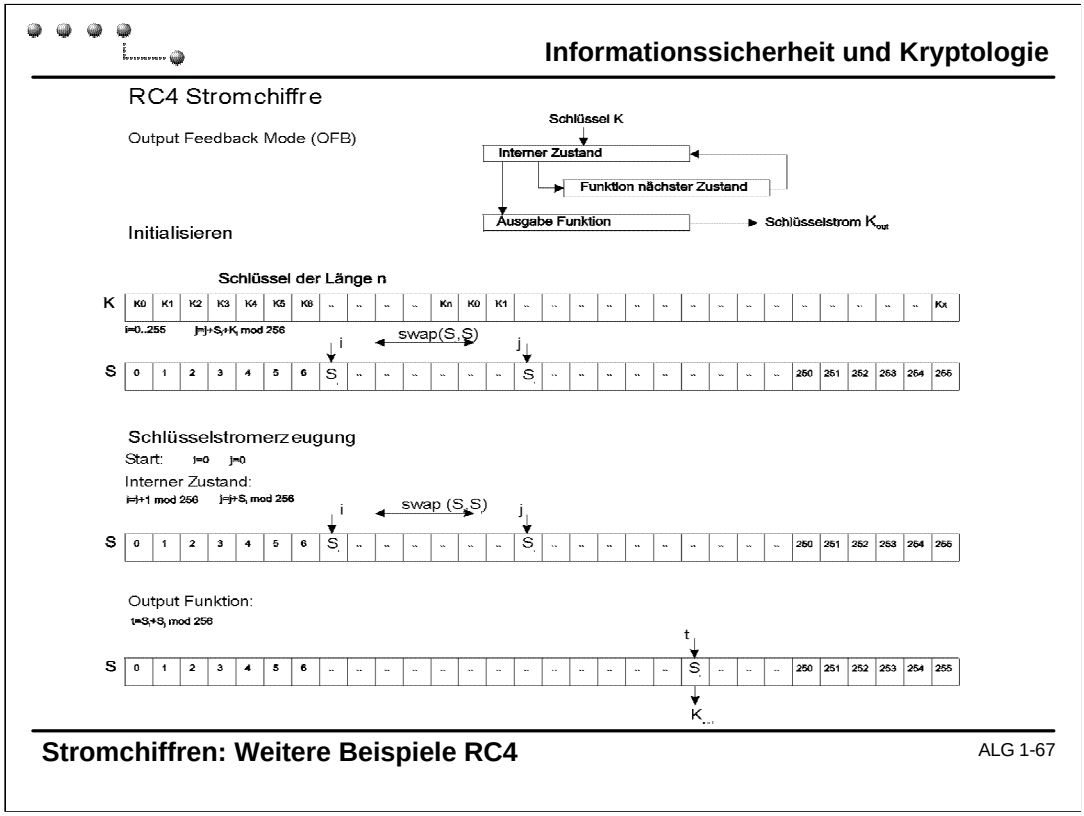
Danach läuft der Generator für 328 Taktzyklen, wobei die letzten 228 Outputbits als Schlüsselstrom für die Verschlüsselung der beiden 114 bit GSM Frames (up-down Nachricht) gebraucht werden. Dieser Prozess wiederholt sich alle 4.6 msec für jedes GSM Frame.

Eine Implementation von A5-1 findet man z.B. unter

http://jya.com/a51-pi.htm

Weitere Informationen findet man in der Linkliste und der weiterführenden Literatur.

A5-1 wird oft nur mit einem Schlüssel der Länge |K|= 54 bit implementiert und er ist deshalb relativ einfach zu knacken, da man im normalen Telefongespräch immer davon ausgehen kann, dass zu Beginn einige 10-tels Sekunden Ruhe ist und dies für einen Klartextangriff ausreicht.



Stromchiffren: Weitere Beispiele RC4 ALG 1-67

Beispiel eines Stromchiffres, der speziell für die Implementierung in Software entwickelt wurde, ist der RC4. RC4 weist eine variable Schlüssellänge auf und hat damit wegen der bis vor kurzem bestehenden US-amerikanischen Exportrestriktionen für Verschlüsselungsverfahren mit Schlüssellängen von über 40 Bit grosse Verbreitung gefunden hat.

Der RC4 arbeitet im Output-Feedback-Modus (OFB), wobei der Schlüsselstrom unabhängig vom Klartext ist.

Es gibt eine S-Box S_0, S_1, ..., S_{255} der Grösse 2^8. Die Einträge sind eine Permutation der Zahlen 0–255, wobei die Permutation von dem Schlüssel variabler Länge abhängt.

Es gibt 2 Zähler; i und j, die mit 0 initialisiert werden. Die Erzeugung eines Zufallsbytes K_{out} verläuft wie folgt:

$i=(i+1) \bmod 256$

$j=(j+S_i) \bmod 256$

vertausche S_i und S_j

$t=(S_i + S_j) \bmod 256$

$K_{out} = S_t$

Das Byte K_{out} wird mit dem Klartext XOR-verknüpft, um den Chiffretext zu erhalten, beziehungsweise mit dem Chiffretext, um den Klartext zu erhalten. Die Verschlüsselung erfolgt etwa 10 mal so schnell wie bei DES.

Die Initialisierung der S-Box ist ebenfalls einfach. Zuerst wird sie linear gefüllt:

S0 = 0, S1 =1, ..., S255 = 255. Dann füllt man ein weiteres Feld der Grösse 256 Byte mit dem Schlüssel K, wobei der Schlüssel so oft wiederholt wird, bis das ganze Feld gefüllt ist:

K0, K1...,K255.

Der Index j wird mit Null belegt. Dann durchläuft man folgende Schleife:

Für i = 0 bis 255

$j = (j + Si + Ki) \bmod 256$

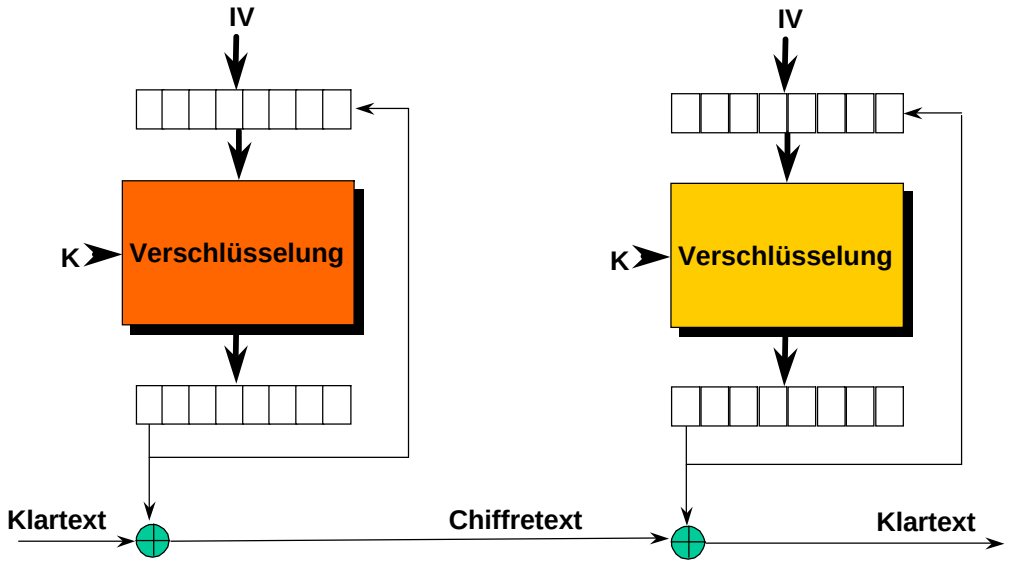
vertausche Si und Sj

RC4 kann etwa 2^{1700} (dh. 256! * 256^2) mögliche Zustände annehmen – eine enorme Zahl.

Der Algorithmus wird von einer ganzen Reihe sehr bekannter Softwareprodukte verwendet. So z.B. in SSL (Netscape, MSIE), Lotus Notes, Windows password encryption und Windows NT SYSKEY, MS Access und PPTP, Adobe Acrobat, Oracle Secure SQL und Apple AOCE, um nur die einige Produkte zu nennen.

Übung: Toy RC4

Der Algorithmus lässt sich natürlich leicht skalieren. Anstelle von Bytes (8-bit) für S können auch kürzere oder längere Worte (2,3, 4 aber auch 16, 32 bit) verwendet werden. Untersuche den Fall |S|=3.



Stromchiffren aus Blockchiffren: Output Feedback (OFB) ALG 1-68

Der OFB-Modus ist eine Betriebsart, in der eine Blockchiffre als Pseudozufallsgenerator für eine Stromchiffre dient. Dazu wird jeweils ein m-Bit Block des Outputs der Verschlüsselungsoperation in das mit einem Initialisierungsvektor vorbesetzte Inputregister zurückgekoppelt. Dieser Teilblock stellt auch das jeweils nächste Glied der Schlüsselfolge dar und wird mit m Bit der Klartextfolge XOR-verknüpft.

Der Wert m kann dabei zwischen 1 und der gesamten Blocklänge frei gewählt werden. Der Modus wird dann auch oft mit OFB-m bezeichnet. Zur Verschlüsselung von m Bit des Klartextes ist in dieser Betriebsart eine Verschlüsselungsoperation nötig, so dass der Durchsatz im allgemeinen (und ganz besonders im Fall m=1) nur einen kleinen Bruchteil des Wertes für den ECB- oder CBC-Modus erreicht.

Für die Implementierung in Software bieten sich auch die sogenannten Kongruenzgeneratoren an, wobei aus einem Anfangswert x_0 der jeweils nächste Wert durch

$$x_i+1 = ax_i + b \bmod m$$

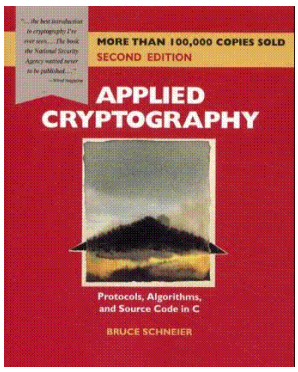
berechnet wird.

Bei richtiger Wahl der Parameter a, b, und m kann man für die höherwertigen Bits der x_i gute statistische Eigenschaften erhalten. Bezüglich der Vorhersage der Folge haben sich die meisten dieser Generatoren jedoch als nicht ausreichend sicher entpuppt, so dass sie heute nicht mehr für kryptographische Anwendungen zum Einsatz kommen sollten.

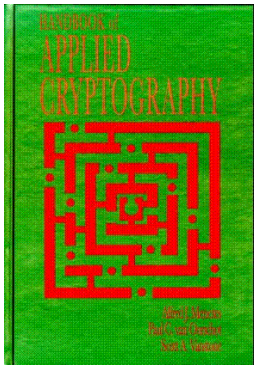
Allgemein gibt es aber nur wenige weit herum bekannte Stromchiffren. Die universeller einsetzbaren Blockchiffren werden dagegen weitaus häufiger diskutiert.

Literaturempfehlungen

Schneier:
Applied Cryptography



Menezes, van Oorschot, Vanstone:
Handbook of Applied Cryptography



Literaturempfehlungen

Schneier: Applied Cryptography,

Wiley, 1996 (2nd edition)

bietet dem Praktiker einen guten und praxisorientierten (fast enzyklopädischen) Überblick der relevanten Verfahren der Kryptologie und C-Code einiger wichtiger Algorithmen.

Das Buch wurde in deutscher Übersetzung unter dem Titel „Angewandte Kryptographie“ herausgegeben, Addison-Wesley, 2000.

Menezes, van Oorschot, Vanstone: Handbook of Applied Cryptography,

CRC Press, 1996

gibt eine umfassende Darstellung der wichtigen Konzepte und Verfahren. Die Darstellung geht wesentlich weiter in die Tiefe als bei Schneier.

Beide werke enthalten ein umfassendes Literaturverzeichnis, mit dessen Hilfe man Originalveröffentlichungen und spezialisierte Literatur leicht auffinden kann.

$x_2\ x_1\ x_0$	x_0	x_1	x_2	$x_0 \oplus x_1$	$x_0 \oplus x_2$	$x_1 \oplus x_2$	$x_0 \oplus x_1 \oplus x_2$
0 0 0	0	0	0	0	0	0	0
0 0 1	1	0	0	1	1	0	1
0 1 0	0	1	0	1	0	1	1
0 1 1	1	1	0	0	1	1	0
1 0 0	0	0	1	0	1	1	1
1 0 1	1	0	1	1	0	1	0
1 1 0	0	1	1	1	1	0	0
1 1 1	1	1	1	0	0	0	1

Anhang: lineare Funktionen von 3 binären Variabeln

Alle linearen booleschen Funktionen mit drei Inputs. Die Komplemente der angegebenen (eigentlich affinen) Funktionen werden in diesem Zusammenhang auch als linear angesehen.