CODING - Stromchiffrier-Methoden mit Komponenten-Änderungen während der Datenbearbeitung

Jürgen Müller, Dipl.-Math.
D-64289 Darmstadt, Deutschland
sysjm(at)t-online(point)de

Kurzbeschreibung

Kern der hier vorgestellten symmetrischen Block-Chiffrierverfahren ist die Kopplung von XOR-Operationen und aus Schlüsseln hergeleiteten, invertierbaren Substitutions-Tabellen **S** mit allen 256**t möglichen Byte-Gruppen (mit t=1, 2, 3,... Bytes, zu Anfang festgelegt):

K(Block) := S(S(Block) ⊗ E_o) ⊗ E_u mit

- E_o obere und E_u untere (Byte-Gruppen-)Dreiecksmatrix mit (Byte-Blocklänge/t)**2 Werten, auf allen nicht Null Positionen mit Wert 1,
- ⊕ die Byte-Gruppen -weise Addition ohne Übertrag ('xor'; 'not xor' ist auch möglich),
- ⊗ die zu ⊕ gehörende (Vektor-)Multiplikation.

Variable Blocklängen (v*t | mod t >0) sind möglich. Der Kern kann n-fach angewandt werden:

K_n(Block) := K(...K(Block)...) mit n K-Operationen, wobei n variabel sein kann.

Da XOR-Operationen und S-Tabellen nur "sinnvoll" funktionieren, wenn 'Block' nicht zu "homogen" ist, und zur Sicherheit, werden aus Schlüsseln zwei weitere Komponenten ermittelt

- Parameter von 2 Pseudo-Zufallszahlen-Prozessen, - Operations-Schlüssel,

die am Anfang und Ende angewandt zu einem chiffrierten Block führen:

cBlock := S(ZZ_2 ⊕ S(Op_E ⊕ S(K_n(Op_A ⊕ S(ZZ_1 ⊕ S(Block)))))) mit

- ZZ_1 und ZZ_2 die Bytes des 1. und 2. Pseudo-Zufallszahlen-Prozesses in Blocklänge,
- Op_A und Op_E der (1./untere und 2./obere Teil oder ein Mehrfaches des) Operations-Schlüssel(s).

Ein Ausgangs-Schlüssel wird zunächst auf t*256**t Bytes erweitert (alle weiteren Schlüssel haben auch diese Größe) und kann so modifiziert werden, dass das Ergebnis sich statistisch nicht von einem Zufallsschlüssel unterscheidet.

Indem man für eine invertierbare S-Tabelle jeweils den Wert (modulo n) nur so vieler fortlaufender Bits eines Schlüssels zur Darstellung der Zahl n-1 ermittelt, um den jeweils die letzten n S-Tabellen-Elemente zyklisch verschoben werden, n=2 bis 256**t, lassen sich alle 256**t! solche S-Tabellen aus den möglichen Schlüssel-Anfängen erzeugen.

Der Byte-Gruppen-Wert +1 an 1. Stelle einer S-Tabelle bestimmt die Byte-Gruppe im Schlüssel, ab der 2*7 Bytes zur Initialisierung zweier Gleitpunktzahlen (IEEE 754) für einen Pseudo-Zufallszahlen-Prozess verwendet werden. Gleitpunktzahlen werden neu belegt, wenn dieser Prozess zyklisch wird.

Idee ist, auch (Operations-)Schlüssel wie Daten-Blöcke zu modifizieren und damit mehr oder weniger "ständig" während der Daten-Chiffrierung neue S-Tabellen, neue Pseudo-Zufallszahlen-Prozesse und neue Operations-Schlüssel zu erzeugen und zu verwenden.

Untersuchungen zeigen, dass trotz Kenntnis von 2 der 3 Komponenten S-Tabelle, Pseudo-Zufallszahlparameter und Operations-Schlüssel sowie Kenntnis von Ausgangs- und chiffrierten Daten nicht auf die fehlende 3. Komponente rückgeschlossen werden kann, falls "gelegentlich" Komponenten-Modifikationen durchgeführt werden.

Ebenfalls zeigt sich, dass durch Kenntnis der 3 Komponenten, die aus einem Schlüssel erzeugt werden, (durch Übergangs-Operations-Schlüssel) nicht auf den Schlüssel selbst rückgeschlossen werden kann. D.h. Kompromittierung von Daten betrifft nicht Daten, die vor dem Komponenten-Wechsel zu den kompromittierten Komponenten chiffriert wurden. Durch Einführung zusätzlicher, separater Komponenten nur für Schlüssel-Modifikationen kann sicher gestellt werden, dass Datenabschnitte, die nach einem Komponenten-Wechsel ausgehend von den kompromittierten Komponenten chiffriert werden, nicht automatisch kompromittiert sind.

Damit sollte eine sichere Stromchiffrierung möglich sein, wie sie bereits für t=1,2,3 umgesetzt wurde.

Einleitung / Zusammenfassung

Kern-Definition

§ Kern der hier vorzustellenden Chiffrier-Methoden ist die Kopplung von XOR Operationen und invertierbaren Substitutions-Tabellen **S** für **Byte-Gruppen** mit **t**=1,2,3,… Bytes (zu Anfang festgelegt):

$$K(Block) := S(S(Block) \text{ Ä } E_o) \text{ Ä } E_u \quad \text{mit}$$

- E_o obere und E_u untere (Byte-Gruppen-) Dreiecksmatrix mit (Byte-Blocklänge/t)^2 Werten, Wert 1 an allen nicht Null Positionen,
- Å Byte-Gruppen-weise Addition ohne Übertrag ('xor'; 'not xor' ist auch möglich),
- Ä die zu ⊕ gehörende (Vektor-) Multiplikation.

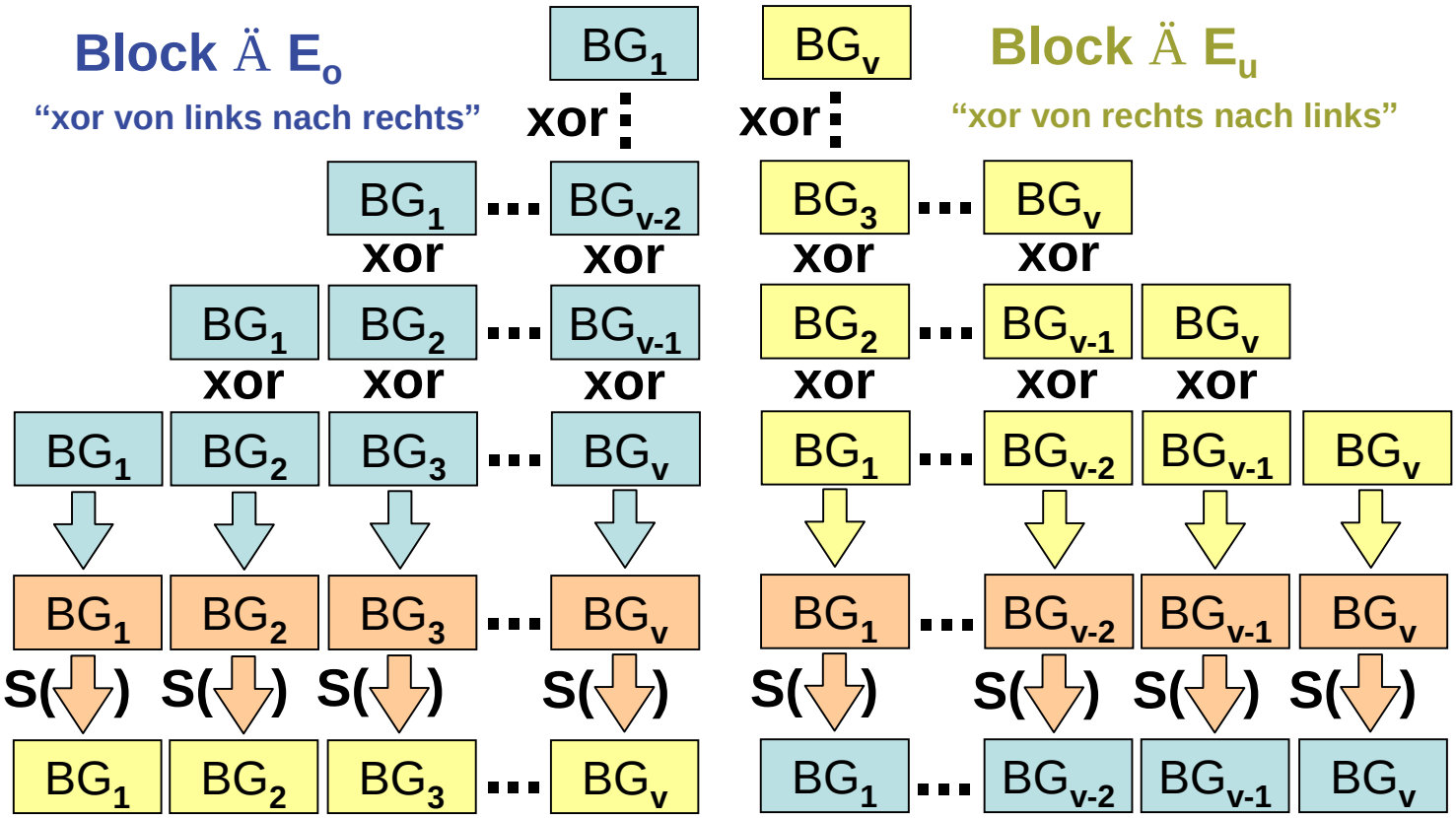
§ Dieser Kern kann **n-fach** angewendet werden:

$$K_n(Block) := K(...K(Block)...) \quad \text{mit}$$

- n K-Operationen, wobei n variable sein kann.

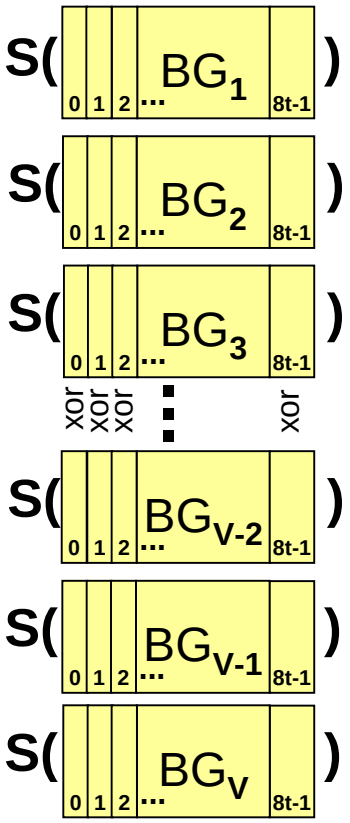
§ Es kann mit variabel langen Blöcken der (Byte-)Länge v*t gearbeitet werden, beim letzten Block auch mit Längen modulo t >0.

Kern-Erklärung (1)



BG_x = Byte-Gruppe x
v = Byte-Gruppen-Anzahl im Block
S(.) = S-Tabellen-Anwendung

Kern-Erklärung (2)



- Die XOR-Operation verknüpft die Bits Spalten-weise je über alle Byte-Gruppen getrennt nach jeder Bit-Stelle.
- Die Anwendung der S-Tabelle modifiziert und „verknüpft“ die Bits zeilenweise pro Byte-Gruppe.
- Die S-Tabelle wird auch zwischen den XOR-Schritten angewandt, da sonst „xor von rechts nach links“ gleich einer Anfangs-Permutation vor „xor von links nach rechts“ ist.
- Genannte Operationen können als eine Art „Bit-Reißwolf“ angesehen werden, in dem alle Bits eines Blocks in Wechselwirkung miteinander treten. Die Kombination sichert, dass jedes Bit eines Blocks auch tatsächlich Einfluss auf jedes andere Bit des betrachteten Blocks nimmt.
- Gemäß dem „Piling-up-Lemma“ konvergieren XOR-verknüpfte stochastisch unabhängige binäre Zufallsvariablen gegen die Gleichverteilung von 0 und 1.
- Werden n Datenmengen mit gleicher aber voneinander unabhängiger Binär-Verteilung **p** per XOR-Operation miteinander verknüpft, so gilt für die Verteilung $\mathbf{p_G}$ der Ergebnisdatenmenge **G**:
  $\left|(p_G-(1-p_G))\right| = \left|(p-(1-p))^n\right| \Rightarrow p_G \hat{I} \{½ \pm ½ * \left|(p-(1-p))^n\right|\}.$
- Beispiel: n=512, p=0,01 $\Rightarrow \left|p_G-½\right| \approx 1{,}61*10^{-5}$
  p=0,001 $\Rightarrow \left|p_G-½\right| \approx 0{,}1794.$

Definition der Chiffrier-Hülle

- Da XOR-Operationen und S-Tabellen nur „sinnvoll“ funktionieren, wenn 'Block' nicht zu „homogen“ ist, und zur Sicherheit, werden aus Schlüsseln zwei weitere Komponenten ermittelt
  - Parameter von 2 Pseudo-Zufallszahlenprozessen,
  - Operations-Schlüssel,

  die am Anfang und Ende angewandt zu einem chiffrierten Block führen:

  cBlock := $S(ZZ_2$ Å $S(Op_E$ Å $S(K_n(Op_A$ Å $S(ZZ_1$ Å $S(Block))))))$ mit
  - **ZZ_1** und **ZZ_2** die Bytes des 1. und 2. Pseudo-Zufallszahlen-Prozesses in Blocklänge,
  - **Op_A** und **Op_E** der (1./untere und 2./obere Teil oder ein Mehrfaches des) Operations-Schlüssel(s).

Schlüssel (1)

- Ein Benutzer-**Schlüssel** wird durch Kopieren zunächst auf **t*256^t Bytes** erweitert, da alle Schlüssel prinzipiell diese Länge besitzen.
- Da das keine „sinnvolle“ Lösung ist, wird eine Methode verwendet, die als **Schlüssel-Überladung** / **-Überlagerung** bezeichnet wird:
  - Wähle Dateien aus, die eine Datenmenge bilden.
  - Gib an, wie oft diese Datenmenge chiffriert werden soll.

  Dann werden jeweils **t Bytes** der chiffrierten Datenmenge als **Positionswert** im Schlüssel und **t Bytes** als **Schlüsselwert** interpretiert und der Ausgangs-Schlüssel entsprechend verändert.
- Es kann gezeigt werden (siehe “Konvergenz von Überlade-Operationen bei Schlüsseln” später) , dass ein **Zufallsschlüssel** mit 256^t Byte-Gruppen im Durchschnitt für t=**1: 61,91%,** t=**2: 61,80%,** t=**3: 61,80%** aller möglichen Byte-Gruppen enthält [**Erwartungswert E(X,t)**].
- Bei **empirisch** untersuchten Überladungen mit den hier vorgestellten Chiffriermethoden ergab sich: t=1: 63,28%, t=2: 63,22%, t=3: 63,21%.

Schlüssel (2)

- Die **Golombschen Axiome** zur Beurteilung von Pseudozufallsfolgen sind „statisch“ und Pseudozufallsfolgen, die diese nachweislich erfüllen, in vielen Fällen „berechenbar“ statt „zufällig“.
  D.h. die Prüfung dieser Axiome liefert nur „erfüllt“ oder „nicht erfüllt“ oder „nicht ermittelbar“.
- Bildet man jedoch aus einer Pseudozufallsfolge (Position,Wert)-Tupel und überlädt damit einen „homogenen“ 256^t Byte-Gruppen langen Schlüssel, dann kann ermittelt werden, wie viel Prozent aller möglichen Byte-Gruppen der überladene Schlüssel im Durchschnitt enthält.
- Die Nähe solcher Werte zu den theoretischen Erwartungswerten kann als **Gütemaß für Zufälligkeit** angesehen werden.
- Die Anzahl der Überladungen ist nicht geeignet, da hier u.a. zunächst ein Signifikanzniveau festgelegt werden müsste, um zu einem Vergleichswert zu kommen (gemäß Untersuchungen in “Konvergenz von Überlade-Operationen bei Schlüsseln” später), usw.

Substitutions-Tabellen S

§ Eine **invertierbare Substitutions-Tabelle S** sei eine Tabelle mit genau allen 256^t möglichen Byte-Gruppen als Tabellen-Elemente.

§ Alle invertierbaren S-Tabellen lassen sich aus den möglichen Werten der **ersten (8t–1)*256^t+1 Schlüssel-Bits** erzeugen:

- S-Tabelle mit allen gemäß Wert aufsteigenden Byte-Gruppen (identische Abbildung) initialisieren oder beliebige, invertierbare S-Tabelle
- Es werden nur zur Darstellung von n-1 nötige fortlaufende Schlüssel-Bits betrachtet, die (mod n) angeben, um wie viel Stellen die letzten n S-Tabellen-Elemente zyklisch verschoben werden sollen, n=2 bis 256^t.

§ *t*: *S-Tabellen-Anzahl* 256^t!: *Schlüssel-Größe* t*256^t: *Schlüsselraum* (256^t)**(256^t):

5/8	10**	35,42	*(in Bytes)*	20	10**	48,16
6/8	10**	89,10		48	10**	115,60
7/8	10**	215,59		112	10**	269,72
1	10**	506,93		256	10**	616,51
2	10**	287.193,71		131.072	10**	315.652,83
3	10**	113.924.438,60		50.331.648	10**	121.210.686,23

Pseudo-Zufallszahlen-Prozesse

§ Aus einem Schlüssel und einer S-Tabelle können die Parameter für einen oder mehrere **Pseudo-Zufallszahlen-Prozesse** ermittelt werden.

§ Dazu wird der Byte-Gruppen-Wert +1 an einer Stelle der S-Tabelle verwendet, um die Byte-Gruppe im Schlüssel zu bestimmen, ab der in den hier vorgestellten Verfahren 2*7 Bytes zur Initialisierung zweier Gleitpunktzahlen (gemäß IEEE 754) verwendet werden.

§ Die Mantissenbytes des Ergebnisses der Multiplikation des Quadrats der 1. mit dem Quadrat der 2. Gleitpunktzahl bilden "Pseudo-Zufallsbytes", während das Quadrat der 1. als neue 2. Gleitpunktzahl zugewiesen und beide Exponenten auf Null gesetzt werden, usw.

§ Gleitpunktzahlen werden neu belegt, wenn ein Prozess zyklisch wird.

§ Zufallsbytes sind für das **„Rauschen“** zuständig, da nicht von inhomo-genen Daten auszugehen ist. Diese aber ermöglichen erst, dass S-Tabellen sowie XOR-Operationen größtmöglich zur Wirkung kommen.

Komponenten-Wechsel (1)

§ Als **Komponenten-Wechsel** wird der Übergang vom Ausgangs-Schlüssel oder von einem **Komponentensatz** bestehend aus
- (invertierte und) S-Tabelle,
- 2 Pseudo-Zufallszahlen-Prozesse,
- Operations-Schlüssel

zum ersten oder nächsten solchen Komponentensatz verstanden.

§ Ausgangspunkt ist der **(Operations-)Schlüssel**, der hierbei **als Datenblock** behandelt wird.

§ Bei der **Erzeugung der Komponenten** werden folgende Schritte ausgeführt:
- **A**. Erzeugung **1. Übergangs-Operations-Schlüssel**
- **B**. Erzeugung **(ggf. inverse und) S-Tabelle**
- **C**. Erzeugung **2. Übergangs-Operations-Schlüssel**
- **D**. Initialisierung der **2 Pseudo-Zufallszahlen-Prozesse**
- **E**. Erzeugung **eigentlicher/Ausgangs-Operations-Schlüssel**

§ wobei **Operations-Schlüssel** (z.B.) wie folgt erzeugt werden:
- **(1)** ‘xor’ Zufallszahlen 1 (ZZ_1; Schlüssel)
- **(2)** S-Tabelle (B für C, D und E; B-Vorgänger für A) anwenden
- **(3)** ‘xor’ der Schlüssel-Daten von links nach rechts
- **(4)** S-Tabelle (B für C, D und E; B-Vorgänger für A) anwenden
- **(5)** ‘xor’ der Schlüssel-Daten von rechts nach links
- **(6)** S-Tabelle (B für C, D und E; B-Vorgänger für A) anwenden
- **(7)** zyklische Verschiebung (gemäß Ergebnis(4) und aktueller Zufallszahl)
- **(8)** Wiederholung der Schritte (3) bis (7)
- **(9)** ‘xor’ der Zufallszahlen 2 (ZZ_2; Schlüssel)
- **(10)** S-Tabelle (B für C, D und E; B-Vorgänger für A) anwenden
- **(11)** Wiederholung der Schritte (3) bis (8) mit (3) und (5) vertauscht.

§ Bei dieser Art des Komponenten-Wechsels kann durch Kenntnis der 3 Komponenten, die aus einem Schlüssel erzeugt werden, **nicht auf den Schlüssel** selbst **rückgeschlossen werden** (siehe Untersuchung in “Kurzuntersuchung zur Sicherheit des Algorithmus” später).

Komponenten-Wechsel (2)

§ Trotz Kenntnis von 2 der 3 Komponenten sowie Kenntnis von Ausgangs- und chiffrierten Daten kann **nicht auf die fehlende 3. Komponente rückgeschlossen werden** (siehe Untersuchungen in “Kurzuntersuchung zur Sicherheit des Algorithmus” später).

§ Dies kann u.a. dadurch gewährleistet werden, dass
- der Operations-Schlüssel höchstens 256^t -1 Mal verwendet wird, bevor dieser durch
  $$\textbf{Operations-Schlüssel}_{neu} := K_n(\textbf{Operations-Schlüssel}_{alt})$$
  geändert wird,
- Gleitpunktzahlen neu belegt werden, falls ein Pseudo-Zufallszahlen-Prozess zyklisch wird,

falls nicht zuvor bereits ein **vollständiger Komponenten-Wechsel** stattfindet, der spätestens mit dem Zyklischwerden des 1. Pseudo-Zufallszahlen-Prozesses initiiert wird (oder nach Behandlung einer festlegbaren durchschnittlichen Anzahl von Daten-Blöcken).

§ D.h. die Kompromittierung eines Komponentensatzes und damit von mit Hilfe dieser Komponenten chiffrierten Daten **betrifft nicht Daten**, die **vor dem Komponenten-Wechsel** zum kompromittierten Komponentensatz chiffriert wurden.

§ Durch Einführung **separater Komponenten nur für die Schlüssel-Chiffrierung** vor dem eigentlichen Komponenten-Wechsel kann sicher gestellt werden, dass **Daten**, die **nach einem** von den kompromittierten Komponenten ausgehenden **Komponenten-Wechsel** chiffriert werden, und die dabei verwendeten Komponenten **nicht** automatisch **kompromittiert** sind.

§ Somit sollte eine **sichere Stromchiffrierung** beliebig umfangreicher Daten für sehr große Zeiträume ohne weiteren Schlüssel- bzw. Komponenten-Austausch möglich sein.

Parameterraum-Größe

§ Allgemein gilt für die **CODING-Parameterraum-Größe**
npars := **nstabs * nkeys * nrannums / delta**
ca.= 10**1.155,40 | 10**602.877,14 | 10**235.135.154,14 für t=1 | 2 | 3
mit

- **nstabs** := S-Tabellen-Anzahl (® siehe Seite 5),
- **nkeys** := Schlüsselraum-Größe (® siehe Seite 5),
- **nrannums** := Zufallszahlenraum-Größe
= 2**(14*8) ca.= 10**33.71,
- **delta** := **nkeys** / **unikeys**
ca.= 10**1,75 | 10**3,11 | 10**4,40 für t=1 | 2 | 3,
- **unikeys** := Anzahl binär gleichverteilter Schlüssel (durch ‚xor' motiviert)
= (**keylen***8)! / (**keylen***4)!^{2},
- **keylen** := Schlüssel-Länge in Bytes (® siehe Seite 5).

Gruppen-Zugehörigkeit

§ Es ist noch zu klären, zu welchen Untergruppen der **symmetrischen Gruppen S(npars)** die Gruppen der Elemente des jeweiligen Parameterraums eines CODING-Verfahrens isomorph sind, d.h. aus welchen **endlichen einfachen Gruppen** sie konstruiert werden können.

Schlüssel-Länge

§ Aussagen zu max. Schlüssel-Längen gehören der Vergangenheit an:

- Die bereits einsetzbaren CODING-Verfahren verwenden **Ausgangs-Schlüssel-Längen bis 50 MB** (genauer: 48*1024^2 Bytes).
- Mit dem CODING-Ansatz lassen sich Verfahren erzeugen, die **beliebige Ausgangs-Schlüssel-Längen** verwenden können
(auch wenn z.Zt. für t>3 wenig "praktikabel").

Zukunftssicherheit

§ Das **„Hase und Igel"-Spiel** (Gebrüder Grimm) zwischen Chiffrierern und Kryptoanalytikern ist dann **beendet**, wenn sich die Parameter-Zyklenlängen der CODING-Verfahren bzgl. (pseudo-)zufälliger Ausgangs-Schlüssel – ggf. nach Verfahrens-Modifikationen – als „ausreichend lang" herausstellen.
Neben Parameter-Änderungen kann immer dann **von** CODING-Verfahren mit **t auf** solche mit **t+1** gewechselt werden, falls t nicht mehr sicher genug erscheint.

Hardware-Box

§ Die Stromchiffrierung könnte z.B. mit **2 Hardware-Boxen** geschehen:
- **Zusammengesetzt** erzeugen sie zunächst - z.B. mit Hilfe der kosmischen Strahlung - einen **echten Zufallsschlüssel**.
- **Danach werden sie getrennt** an die Stellen bewegt, an denen der Datenstrom ver- und entschlüsselt werden soll.
- Mit dem zur Zeit „sichersten“ CODING-Verfahren (t=3) wäre der **Datenstrom** zwischen den Boxen für beliebige Organisationen mit beliebiger auch zukünftiger Infrastruktur **auf absehbare Zeit nicht Klartext-lesbar**.

Durchsatz

§ Aktuell kann das z.Zt. mittlere CODING-Verfahren (**t=2**) auf einem 2-Prozessor-PC (x64-System, Intel i7 CPU, 2,67MHz, 8192MB Core) mit ca. **20 MB/sec Durchsatz** chiffrieren, für **t=3** ist der Durchsatz z.Zt. weit kleiner und wäre vor Einsatz ggf. **weiter zu optimieren**.

Konvergenz von Überlade-Operationen bei Schlüsseln

Ausgangslage

Bei der Überladung von Schlüsseln ergibt sich sehr schnell die Frage, wie viele Überladungen stattfinden müssen, damit der daraus resultierende veränderte Schlüssel nicht mehr von einem „zufälligen“ Schlüssel zu unterscheiden ist. Aber was ist ein „zufälliger“ Schlüssel ?

Ein Schlüssel habe **S Stellen** in der **Stellenmenge *S*** und **W unterschiedliche Werte** in der **Wertemenge *W***, W≤S und **w(s)** sei der **Wert w** an der **Stelle s** im Schlüssel. Sei **Z ein Zufallsprozess** für **Tupel T**:=**(s(Z),w(Z))** aller möglichen Werte und Stellen mit einzelnen Werten **w(Z)** und Stellen **s(Z)**, wobei w(Z), s(Z) unabhängig von einander seien. Sei **W**=**n** die Anzahl von *W* bzw. **#(w)**, w∈ *W* die Anzahl des gleichen Wertes w bezogen auf alle Stellen *S*.

Mit jedem Tupel (s(Z),w(Z)) werde der Wert w an der Stelle s(Z) im Schlüssel durch den Wert w(Z) ersetzt: **w(s(Z))** := w(Z) . Dementsprechend bezeichnen $\boldsymbol{W_{alt}}$ bzw. $\boldsymbol{W_{neu}}$ die Menge *W* vor und nach der Ersetzung bzw. $\mathbf{W_{alt}}$ bzw. $\mathbf{W_{neu}}$ deren unterschiedliche Werteanzahl.

1. Aufgabe: Um welchen Anzahl-Wert W=N schwankt der Schlüssel für einen Zufallsprozess Z ?

Es gilt:

P(eine Stelle s(Z) aus Z) = 1/S,
P(ein Wert w(Z) aus Z) = 1/S,
$P(w(Z)\in W) = P(w(Z)=w_1\in W) + P(w(Z)=w_2\in W) +...+ P(w(Z)=w_n\in W) = n/S$,
$P(w(Z)\notin W) = (S-n)/S$.

Falls $w(Z)\notin W$, dann gilt:

$w(s(Z))=w\ [\in W] \wedge \#(w)=1 \Rightarrow W_{neu} = W_{alt}$,
$w(s(Z))=w\ [\in W] \wedge \#(w)\neq 1 \Rightarrow W_{neu} = W_{alt} +1$.

Falls $w(Z)\in W$, dann gilt:

$w(s(Z))=w\ [\in W] \wedge \#(w)\neq 1 \Rightarrow W_{neu} = W_{alt}$,
$w(s(Z))=w\ [\in W] \wedge \#(w)=1 \wedge w(Z)=w(s(Z)) \Rightarrow W_{neu} = W_{alt}$,
$w(s(Z))=w\ [\in W] \wedge \#(w)=1 \wedge w(Z)\neq w(s(Z)) \Rightarrow W_{neu} = W_{alt} -1$.

Daher sind folgende Werte zu ermitteln:

$P(W_{neu} = W_{alt})$,
$P(W_{neu} = W_{alt} +1)$,
$P(W_{neu} = W_{alt} -1)$.

Sei daher **D** die **Anzahl der Mehrfach-Werte *D*** von *W*, d.h. $w\in D \Rightarrow \#(w)>1$, D≤W und $\mathbf{S_D}$ die **Anzahl der Stellen S_D mit Mehrfach-Werten** *D*. Dann gilt:

$S-n \le S_D \le 2*(S-n)$, falls $n \ge S/2$
$\le S$, sonst.

$S-S_D$ ist die Anzahl der Stellen mit Werten $w\in W$, #(w)=1, d.h. aber $w\in W-D$, somit gilt:

$S - S_D = W - D \Rightarrow S = n + S_D - D \Leftrightarrow D = n + S_D - S$,
$P(s(Z)\in S_D) = S_D/S$,
$P(s(Z)\notin S_D) = (S-S_D)/S$.

Daraus folgt unter der Annahme eines festen n- und S_D-Wertes (n,S_D):

$P(W_{neu}=W_{alt}$ für $(n= W_{alt},S_D))$
$= P(w(Z)\notin W \wedge w(s(Z))=w \wedge \#(w)=1)$
$+ P(w(Z)\in W \wedge w(s(Z))=w \wedge \#(w)\neq 1)$
$+ P(w(Z)\in W \wedge w(s(Z))=w \wedge \#(w)=1 \wedge w(Z)=w(s(Z)))$

$= P(w(Z)\notin W \wedge s(Z)\notin S_D)$
$\quad + P(w(Z)\in W \wedge s(Z)\in S_D)$
$\quad + P(w(Z)\in W \wedge s(Z)\notin S_D \wedge w(Z)=w(s(Z)))$
$= (S-n)/S * (S-S_D)/S + n/S * S_D/S + n/S * (S-S_D)/S * (S-S_D)/(n*(S-S_D))$
$= (S^2-S*(n+S_D)+n*S_D)/S^2 + n*S_D/S^2 + (S-S_D)/S^2$
$= (S^2-S*n-S*S_D+n*S_D+n*S_D+S-S_D)/S^2$
$= 1 -(n-1)/S + S_D*(2*n-S-1)/S^2$

Für n=S ist S_D=0 und damit $\Rightarrow$ =1/S (**P**).
Für S_D=S $\Rightarrow$ jeder Wert liegt mehrfach vor $\Rightarrow$ =n/S (**P**).

$P(W_{neu}=W_{alt}+1 \text{ for } (n= W_{alt},S_D))$
$= P(w(Z)\notin W \wedge w(s(Z))=w \wedge \#(w)\neq 1)$
$= P(w(Z)\notin W \wedge s(Z)\in S_D)$
$= (S-n)/S * S_D/S$
$= S_D*(S-n)/S^2$

Für n=S $\Rightarrow$ =0 (**P**).
Für S_D=S $\Rightarrow$ jeder Wert liegt mehrfach vor $\Rightarrow$ =1−n/S (**P**).

$P(W_{neu}=W_{alt}-1 \text{ für } (n=W_{alt},S_D))$
$= P(w(Z)\in W \wedge w(s(Z))=w \wedge \#(w)=1 \wedge w(Z)\neq w(s(Z)))$
$= P(w(Z)\in W \wedge s(Z)\notin S_D \wedge w(Z)\neq w(s(Z)))$
$= n/S * (S-S_D)/S * (1 - (S-S_D)/(n*(S-S_D)))$
$= n*(S-S_D)/S^2 - (S-S_D)/S^2$
$= (n-1)*(S-S_D)/S^2$

Für n=S ist S_D=0 und damit $\Rightarrow$ =1−1/S (**P**).
Für S_D=S $\Rightarrow$ =0 (**P**).

und

$P(W_{neu}=W_{alt} \text{ für } (n,S_D)) + P(W_{neu}=W_{alt}+1 \text{ für } (n,S_D)) + P(W_{neu}=W_{alt}-1 \text{ für } (n,S_D))$
$= 1 -(n-1)/S + S_D*(2*n-S-1)/S^2 + S_D*(S-n)/S^2 + (n-1)*(S-S_D)/S^2$
$= [S^2 -n*S+S +2*n*S_D-S_D*S-S_D+S_D*S-n*S_D +n*S-n*S_D-S+S_D] /S^2$
$= [S^2+(nS-nS)+(2*n*S_D-n*S_D-n*S_D)+(S_D*S-S_D*S)+(S-S)+(S_D-S_D)] /S^2 =1$ (**P**)

Ausgestaltung:

$P(W_{neu}=W_{alt} +1 \text{ für } S>n\geq S/2)$
$= \sum_{s\in S_D} P(W_{neu}=W_{alt}+1 \text{ für } (n,s))$

à So fehlt die Gewichtung gemäß der Anzahl der potentiell möglichen S_D-Varianten !! **ß**

Es gilt (mit (1)):
Sei S_D=S−n+k: **Gewichtung Gw(S_D)** := $\binom{S-n-1}{k-1} * \binom{n}{k}$ mit $\binom{S-n-1}{k-1}$=1 für k=1.

Normierung:
Norm := $\sum_{j=1,...,(S-n)} \binom{S-n-1}{j-1} * \binom{n}{j} =^{(2)} \binom{S-1}{n-1}$

da $\sum_{s\in S_D} Gw(s)/Norm = \sum_{S-n<s\leq 2*(S-n)} Gw(s)/Norm$
$= \sum_{k=1,...,(S-n)} \binom{S-n-1}{k-1} * \binom{n}{k} /Norm =^{(2)} \binom{S-1}{n-1} / \binom{S-1}{n-1} = 1$.

Daraus ergibt sich:

$$P(W_{neu}=W_{alt}+1 \text{ für } S>n\geq S/2)$$
$$= \sum_{s\in S_D} P(W_{neu}=W_{alt}+1 \text{ für } (n,s))*Gw(s)/Norm$$
$$= \sum_{S-n<s\leq 2*(S-n)} s*(S-n)/S^2 *Gw(s)/Norm \Rightarrow^{(3)}$$
$$= (S+n-1)*(S-n)^2/[S^2*(S-1)]$$

$$P(W_{neu}=W_{alt}-1 \text{ für } S>n\geq S/2)$$
$$= \sum_{s\in S_D} P(W_{neu}=W_{alt}-1 \text{ für } (n,s))*Gw(s)/Norm$$
$$= \sum_{S-n<s\leq 2*(S-n)} (n-1)*(S-s)/S^2 *Gw(s)/Norm \Rightarrow^{(4)}$$
$$= n*(n-1)^2/[S^2*(S-1)]$$

$$P(W_{neu}=W_{alt} \text{ für } S>n\geq S/2)$$
$$= \sum_{s\in S_D} P(W_{neu}=W_{alt} \text{ für } (n,s))*Gw(s)/Norm$$
$$= \sum_{S-n<s\leq 2*(S-n)} (1-(n-1)/S+s*(2*n-S-1)/S^2) *Gw(s)/Norm \Rightarrow^{(5)}$$
$$= [n*(S-n+1)*(S-1)+2*n*(n-1)*(S-n)]/[S^2*(S-1)]$$

Zur Probe der Korrektheit der Werte siehe auch [(6)] unten.

Einfache Gleichgewichtsannahme:

$$P(W_{neu}=W_{alt}+1 \text{ für } S>n\geq S/2) = P(W_{neu}=W_{alt}-1 \text{ für } S>n\geq S/2) \Rightarrow^{(7)}$$
$$0 = n^2 + n*(S-1) - S^2 \Rightarrow$$

für S=256:	$n \approx 158{,}5$	$\rightarrow$ 61,911501555 %
für S=65536:	$n \approx 40503{,}75$	$\rightarrow$ 61,803817749 %
für S=16777216:	$n \approx 10368890$	$\rightarrow$ 61,803400517 %

Dass diese Annahme zum Erfolg führt, lässt sich auch durch folgenden Ansatz belegen:

Erweitertes System:

Gesucht wird der Wert N mit der maximalen Wahrscheinlichkeit: $N := \max_{n=1,...,S}[P(W=n)]$.

Dabei sind die P(W=n) aus folgendem Gleichungssystem zu berechnen:

$$P(W=n) = P(W=n)*P(W_{neu}=W_{alt},W=n) + P(W=n-1)*P(W_{neu}=W_{alt}+1,W=n-1) + P(W=n+1)*P(W_{neu}=W_{alt}-1,W=n+1), \quad 1<n<S,$$
$$P(W=1) = P(W=1)*P(W_{neu}=W_{alt},W=1) + P(W=2)*P(W_{neu}=W_{alt}-1,W=2), \quad (n=1),$$
$$P(W=S) = P(W=S)*P(W_{neu}=W_{alt},W=S) + P(W=S-1)*P(W_{neu}=W_{alt}+1,W=S-1), \quad (n=S),$$
$$\sum_{n=1,...,S} P(W=n) = 1.$$

Die ersten n Gleichungen sind nicht unabhängig, so dass eine davon unberücksichtigt bleibt und durch die letzte ersetzt wird.

Die beispielhaften und einfachen Berechnungen für S=2 siehe [(8)] und für S=3 siehe [(9)].

Allgemein sei für alle n=1,...,S:

$x_n := P(W=n)$,

$a_n := P(W_{neu}=W_{alt},W=n) - 1$
$= [n*(S-n+1)*(S-1) +2n*(n-1)*(S-n)]/[S^2*(S-1)] - 1$,

$p_n := P(W_{neu}=W_{alt}+1,W=n)$
$= (S+n-1)*(S-n)^2/[S^2*(S-1)]$,

$m_n := P(W_{neu}=W_{alt}-1,W=n)$.
$= n*(n-1)^2/[S^2*(S-1)]$

Dann gilt [zum Ansatz mit Determinanten siehe [10]]:

$$\begin{aligned}
x_1+ \quad x_2+ \quad x_3+ \quad x_4+ \ldots + \quad x_{n-2}+ \quad x_{n-1}+ \quad x_n &= 1\\
a_1*x_1+m_2*x_2 &= 0\\
p_1*x_1+ a_2*x_2+m_3*x_3 &= 0\\
p_2*x_2+ a_3*x_3+m_4*x_4 &= 0\\
\ldots &\ldots\\
p_{n-2}*x_{n-2}+a_{n-1}*x_{n-1}+m_n*x_n &= 0\\
p_{n-1}*x_{n-1}+ a_n*x_n &= 0
\end{aligned}$$

oder

$$\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & \ldots & 1 & 1 & 1 & 1\\
p_1 & a_2 & m_3 & 0 & 0 & \ldots & 0 & 0 & 0 & 0\\
0 & p_2 & a_3 & m_4 & 0 & \ldots & 0 & 0 & 0 & 0\\
0 & 0 & p_3 & a_4 & m_5 & \ldots & 0 & 0 & 0 & 0\\
0 & 0 & 0 & p_4 & a_5 & \ldots & 0 & 0 & 0 & 0\\
\ldots & \ldots & \ldots & \ldots & \ldots & & \ldots & \ldots & \ldots & \ldots\\
0 & 0 & 0 & 0 & 0 & \ldots & p_{n-3} & a_{n-2} & m_{n-1} & 0\\
0 & 0 & 0 & 0 & 0 & \ldots & 0 & p_{n-2} & a_{n-1} & m_n\\
0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & p_{n-1} & a_n
\end{pmatrix} * \begin{pmatrix} x_1\\ x_2\\ x_3\\ x_4\\ x_5\\ \ldots\\ x_{n-2}\\ x_{n-1}\\ x_n \end{pmatrix} = \begin{pmatrix} 1\\ 0\\ 0\\ 0\\ 0\\ \ldots\\ 0\\ 0\\ 0 \end{pmatrix}$$

kurz $C * x = b$.

Daraus lässt sich eine obere Dreiecksmatrix erzeugen, deren $n*(n+1)/2$ Elemente sich bekanntlich via Index-Transformation

für $i \le j$: $[i,j] \rightarrow [(i-1)*n-(i-2)*(i-1)/2+j-(i-1)] = [(i-1)*(2*n-i)/2+j]$

in ein 1-dimensionales Array überführen lassen.

Bei genauerer Betrachtung ergib sich aber folgendes System:

$$\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & \ldots & 1 & 1 & 1 & 1\\
0 & \underline{a}_2 & \underline{m}_3 & \underline{r}_2 & \underline{r}_2 & \ldots & \underline{r}_2 & \underline{r}_2 & \underline{r}_2 & \underline{r}_2\\
0 & 0 & \underline{a}_3 & \underline{m}_4 & \underline{r}_3 & \ldots & \underline{r}_3 & \underline{r}_3 & \underline{r}_3 & \underline{r}_3\\
0 & 0 & 0 & \underline{a}_4 & \underline{m}_5 & \ldots & \underline{r}_4 & \underline{r}_4 & \underline{r}_4 & \underline{r}_4\\
0 & 0 & 0 & 0 & \underline{a}_5 & \ldots & \underline{r}_5 & \underline{r}_5 & \underline{r}_5 & \underline{r}_5\\
\ldots & \ldots & \ldots & \ldots & \ldots & & \ldots & \ldots & \ldots & \ldots\\
0 & 0 & 0 & 0 & 0 & \ldots & 0 & \underline{a}_{n-2} & \underline{m}_{n-1} & \underline{r}_{n-2}\\
0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & \underline{a}_{n-1} & \underline{m}_n\\
0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 & \underline{a}_n
\end{pmatrix} * x = \begin{pmatrix} 1\\ \underline{b}_2\\ \underline{b}_3\\ \underline{b}_4\\ \underline{b}_5\\ \ldots\\ \underline{b}_{n-2}\\ \underline{b}_{n-1}\\ \underline{b}_n \end{pmatrix},$$

wobei

$\underline{r}_1 = \underline{m}_2 = \underline{a}_1 = \underline{b}_1 = 1$,
$\underline{r}_i = 0 - \underline{r}_{n-1}*p_{i-1}/\underline{a}_{i-1}$, $i=2,...,n-2$,
$\underline{a}_i = a_i - \underline{m}_i*p_{i-1}/\underline{a}_{i-1}$, $i=2,...,n$,
$\underline{m}_i = m_i - \underline{r}_{n-2}*p_{i-2}/\underline{a}_{i-2}$, $i=3,...,n$,
$\underline{b}_i = b_i - \underline{b}_{i-1}*p_{i-1}/\underline{a}_{i-1}$, $i=2,...,n$,

und zur Vervollständigung sei

$\underline{r}_{n-1} = \underline{r}_n = \underline{m}_{n+1} = 0$.

Es müssen daher nur die 4–Tupel–Werte

$(t_{1,i}, t_{2,i}, t_{3,i}, t_{4,i}) := (\underline{a}_i, \underline{m}_{i+1}, \underline{r}_i, \underline{b}_i)$ für $i=1,...,n$

gespeichert und modifiziert werden, um die Werte der Dreiecksmatrix zu halten, die eingehenden Werte p_i, a_i, m_i und b_i sind ja direkt berechen- bzw. ermittelbar.

Damit können die x_i für i=n bis 1 wie folgt berechnet werden:

$$x_i = [\underline{b_i} - \underline{r_i}*[x_{i+2}+...+x_n] - \underline{m_{i+1}}*x_{i+1}] / \underline{a_i},$$

nur für i<n−1 ◄ ► nur für i<n

oder

$$x_i = [t_{4,i} - t_{3,i}*[x_{i+2}+...+x_n] - t_{2,i}*x_{i+1}] / t_{1,i} .$$

Führt man nun eine Berechnung der Werte N mit Hilfe eines Programms durch, so ergibt sich

für S=256: $N \approx 158$ $\rightarrow 61{,}71875\%$
für S=65536: $N \approx 40504$ $\rightarrow 61{,}804199219\%$
für S=16777216: $N \approx 10368890$ $\rightarrow 61{,}803400517\%$

d.h. die Methode der einfachen Gleichgewichtsannahme wird voll bestätigt.

In vom Autor untersuchten konkreten Fällen lagen die Werte bei ca. 63,28% bzw. 63,22% bzw. 63,21% (siehe [14], [15] und [16] unten). Diese Abweichungen von den hier theoretisch berechneten Werten sind ggf. noch genauer zu untersuchen.

2. Aufgabe: Wie viele Tupel T müssen via Zufallszahlenprozess Z generiert und auf den Schlüssel angewendet werden, bis P(W=N) sicher erreicht wird ?

- Es müssen mindestens $N-N_0$ Tupel sein, falls $\mathbf{N_0}$ die **Anzahl der** bereits **im Ausgangsschlüssel enthaltenen unterschiedlichen Zeichen(-Gruppen)** ist und $N_0<N$.
- Der Begriff „sicher erreicht“ ist genauer zu fassen, [da P(W=N) für $N_0<N$ kleiner 1 sein muss] z.B. mit Hilfe eines **Signifikanzniveaus s** von z.B. s=95%.

Setzt man

$P(W_{neu}=W_{alt}=N) := 1,$
$P(W_{neu}=W_{alt}-1,W_{old}=N) := 0,$
$P(W_{neu}=W_{alt}+1,W_{old}=N) := 0,$

dann gilt allgemein für n>N

$P(W_{neu}=W_{alt}=n) := 0,$
$P(W_{neu}=W_{alt}-1,W_{alt}=n) := 0,$
$P(W_{neu}=W_{alt}+1,W_{alt}=n) := 0$

und das Gesamtsystem verbleibt im Zustand N, falls dieser erst einmal erreicht ist.

Falls die folgenden P die Wahrscheinlichkeiten sind,

- $\mathbf{P_i(n)}$ sich nach der n-ten Tupel-Ersetzung im Zustand W=i zu befinden,
- $\mathbf{P_{i-1,i}}$ mit einer Tupel-Ersetzung vom Zustand W=i−1 zum Zustand W=i zu wechseln [p_{i-1} von oben],
- $\mathbf{P_{i,i-1}}$ mit einer Tupel-Ersetzung vom Zustand W=i zum Zustand W=i−1 zu wechseln [m_i von oben],
- $\mathbf{P_{i,i}}$ mit einer Tupel-Ersetzung im Zustand W=i zu bleiben [$\mathbf{á_i} := a_i+1$ von oben],

dann gilt

$P_1(n) := P_1(n-1)*P_{1,1} + P_2(n-1)*P_{2,1}$
$P_i(n) := P_{i-1}(n-1)*P_{i-1,i} + P_i(n-1)*P_{i,i} + P_{i+1}(n-1)*P_{i+1,i}$
$P_{N-1}(n) := P_{N-2}(n-1)*P_{N-2,N-1} + P_{N-1}(n-1)*P_{N-1,N-1}$
$P_N(n) := P_{N-1}(n-1)*P_{N-1,N} + P_N(n-1)*P_{N,N}$ (wobei $P_{N,N}=1$)

oder mit

$$B := \begin{pmatrix} P_{1,1} & P_{2,1} & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ P_{1,2} & P_{2,2} & P_{3,2} & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & P_{2,3} & P_{3,3} & P_{4,3} & 0 & \dots & 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots & P_{N-3,N-2} & P_{N-2,N-2} & P_{N-1,N-2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & P_{N-2,N-1} & P_{N-1,N-1} & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & P_{N-1,N} & 1 \end{pmatrix} \text{ und}$$

$$P(n) := [P_1(n)\ P_2(n)\ \dots\ P_{N-1}(n)\ P_N(n)]^T$$

gilt

$$P(n) = B*P(n-1) \ \Rightarrow\ P(n) = (B)^n * P(0)$$

Als Beispiel für N=3 siehe [(11)].

Falls P(0) also mit $i=N_0$ an der i-ten Stelle eine 1 aufweist, d.h. $P_i(0)=1$, und $b_{N,i}(n)$ der Wert in der N-ten Zeile und der i-ten Spalte der Matrix $(B)^n$ ist, dann wird der kleinste Wert n gesucht, für den gilt: $b_{N,i}(n) \geq s$.

Dann ist #(T)=n der gesuchte Tupel-Anzahlwert zum Signifikanzniveau s.

Die hier vorliegende Matrix P ist eine stochastische Matrix, d.h. eine Übergangsmatrix einer (endlichen) Markoff-Kette mit absorbierendem Zustand N [wobei hier Zeilen und Spalten vertauscht vorliegen].

Betrachten wir die Gegebenheiten genauer. Es gilt

$$B := \begin{pmatrix} á_1 & m_2 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ p_1 & á_2 & m_3 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & p_2 & á_3 & m_4 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & p_3 & á_4 & m_5 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & p_4 & á_5 & \dots & 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots & p_{n-3} & á_{n-2} & m_{n-1} & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & p_{n-2} & á_{n-1} & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & p_{n-1} & 1 \end{pmatrix}$$

Allgemein lässt sich für beliebige n ermitteln:

$$b_{i,j}(n) = \sum_{k=1,\dots,N}[b_{i,k}(n-1)*b_{k,j}(1)]$$
$$= b_{i,j-1}(n-1)*b_{j-1,j}(1) + b_{i,j}(n-1)*b_{j,j}(1) + b_{i,j+1}(n-1)*b_{j+1,j}(1)$$
$$= b_{i,j-1}(n-1)*m_j + b_{i,j}(n-1)*á_j + b_{i,j+1}(n-1)*p_j, \qquad i=1,\dots,N,\ j=1,\dots,N-1,$$
$$b_{i,N}(n) = 0, \quad i=1,\dots,N-1,$$
$$b_{N,N}(n) = 1\ .$$

Für spezielle n<N gilt (siehe auch [(12)]):

$$b_{i,j}(n) = 0, \qquad \text{falls } |j-i|>n \text{ für alle } n<N,$$
$$b_{i,i+n}(n) = \prod_{k=i+1,\dots,i+n} m_k \qquad \text{für alle } i+n<N,$$
$$b_{i+n,i}(n) = \prod_{k=i,\dots,i+n-1} p_k \qquad \text{für alle } i+n\leq N,$$
$$b_{i,j}(n) = \prod_{k=i+1,\dots,j}[m_k] * \sum_{k=i,\dots,j}[á_k], \qquad \text{falls } j=i+n-1, \text{ für alle } 1<n<N-i-1,$$
$$b_{i,j}(n) = \prod_{k=j,\dots,i-1}[p_k] * \sum_{k=j,\dots,i}[á_k], \qquad \text{falls } i=j+n-1, \text{ für alle } 1<n\leq N-j+1,$$
$$b_{i,j}(n) = \prod_{k=i+1,\dots,j}[m_k] * (\sum_{k=i-1,\dots,j}[p_k*m_{k+1}] + \sum_{k=i,\dots,j}[á_k * \sum_{h=k,\dots,j}[á_h]]), \qquad \text{falls } j=i+n-2, \text{ für alle } 1<n<N-i+2,$$
$$b_{i,j}(n) = \prod_{k=j,\dots,i-1}[p_k] * (\sum_{k=j,\dots,i+1}[m_k*p_{k-1}] + \sum_{k=j,\dots,i}[á_k * \sum_{h=k,\dots,i}[á_h]]), \qquad \text{falls } i=j+n-2, \text{ für alle } 1<n\leq N-j+2.$$

Da P(n) mit $P(n) = (B)^n * P(0)$ beschrieben werden kann und nur die $b_{N,i}(n)$ für ein festes i von Interesse sind, ist von der Matrix $(B)^n$ lediglich die letzte (N.) Zeile von Interesse.

Des weiteren ergibt sich (siehe [13]), dass $b_{N,i} < b_{N,k}$,falls $i<k$ und $b_{N,k} \neq 0$. Falls daher die $b_{N,1}(n)$ betrachtet werden, so können die $b_{N,i}(n)$ mit $i>1$ höchstens größer sein bzw. werden sich für große N und "kleinen" Abständen zu 1 sogar im nahen Bereich von $b_{N,1}(n)$ [=: **P(1®N|n)**] aufhalten.

Führt man nun eine Berechnung für die Werte N mit Hilfe eines entsprechenden Programms durch, so ergeben sich die kleinsten n mit $P(1 \rightarrow N|n) > s$ für

(in %) s=	**S=256, N=158: n=**	**S=65536, N=40504: n=**	**S=16777216, N=10368890: n=**
1,0000	311	163050	82951120(?)
75,0000	541	224514	125256190(?)
90,0000	641	251873	181907150(?)
95,0000	716	272313	?
99,0000	889	319656	?
99,9000	1137	387364	?
99,9900	1384	455070	?
99,9990	1632	522777	?
99,9999	1880	590484	?
empirisch ermittelt: [siehe [14], [15], [16], unten]	nV=5000 (N=162):	nV=800000 (N=41435):	nV*=266500000 (N=10604723):
	n=705 (mZH=8)	n=360363 (mZH=151)	n=142481560 (mZH=1772)
	n=781 (mZH=6)	n=397529 (mZH=87)	n=184027844 (mZH=150)
	n=939 (mZH=4)	n=496542 (mZH=20)	n=212320550 (mZH=30)
	n=1571 (mZH=3)	n=632086 (mZH=7)	n=234952796 (mZH=9)

wobei

nV := Anzahl der (einzeln) betrachteten Überladevorgänge,
nV* := Anzahl der betrachteten Überladevorgänge, wobei nur alle 1066 Überladungen Statistiken erhoben wurden, die zur Berechnung der übrigen Werte dienten,
mZH := min(max(Zeichgruppen-Häufigkeit)) mit n, letzter mZH-Wert bis nV bzw. nV*,
(N=...) := aus den nV bzw. nV* ermittelter Wert N.
...(?) geschätzter Wert; die Programm-interne Werte-Darstellung (Mantissen-Genauigkeit) sowie die Rechner-Geschwindigkeit (PC des Autors) reichte für S=16777216 nicht aus, um in akzeptabler Zeit zu einem Ergebnis selbst für s=1% zu kommen (!).

Erläuterungen:

Aus Vereinfachungsgründen wird $\binom{n}{k}$ im Folgenden auch mit „**(n über k)**“ bezeichnet.

(1) $S_D=S-n+1$: Genau ein Wert wird wiederholt, wobei n Werte in Frage kommen:
Gewichtung =n.

$S_D=S-n+2$: Genau 2 Werte werden wiederholt, wobei n(n−1) in Frage kommen, wovon der eine $x_1=2$ bis $x_1=S-n$ Mal wiederholt wird und der 2. entsprechend $x_2=S-n+2-x_1$ Mal. Da egal ist, welches der 1. und welches der 2. Wert ist, gilt:
Gewichtung =n!/(2!(n−2)!)(S−n−1).

$S_D=S-n+3$: Genau 3 Werte werden wiederholt, wobei n(n−1)(n−2) in Frage kommen, wovon der 1. $x_1=2$ bis $x_1=S-n-1$ (=S−n+3−2∗2) Mal wiederholt wird, der 2. $x_2=2$ bis $x_2=S-n+1-x_1$ (=S−n+3−1∗2−x_1) Mal, der 3. entsprechend $x_3=S-n+3-x_1-x_2$:
Gewichtung =n!/(3!(n−3)!)∗[(S−n−2)+(S−n−3)+...+1]
=(n über 3)(S−n−2)((S−n−2)+1)/2.
=(n über 3)((S−n−2)+1)∗(S−n−2)/(1∗2).

$S_D=S-n+4$: Genau 4 Werte werden wiederholt, wobei n(n−1)(n−2)(n−3) in Frage kommen, wovon der 1. $x_1=2$ bis $x_1=S-n-2$ (=S−n+4−3∗2) Mal wiederholt wird, der 2. $x_2=2$ bis $x_2=S-n-x_1$ (=S−n+4−2∗2−x_1) Mal, der 3. $x_3=2$ bis $x_3=S-n+2-x_1-x_2$ (=S−n+4−1∗2−x_1−x_2) Mal, der 4. entsprechend $x_4=S-n+4-x_1-x_2-x_3$:
Gewichtung =n!/(4!(n−4)!)∗[(S−n−3)(S−n−2)/2+(S−n−4)(S−n−3)/2+...+1∗2/2]
mit $1*2+2*3+3*4+...+n*(n+1) = 1/3*n^3+n^2+2/3*n$
=(n über 4)[$(S-n-3)^2$+3(S−n−3)+2]∗(S−n−3)/(2∗3).
=(n über 4)((S−n−3)+2)∗((S−n−3)+1)∗(S−n−3)/(1∗2∗3).

...

$S_D=S-n+(S-n)=2(S-n)$ [n≥S/2]:
Genau S−n Werte werden wiederholt, wobei n(n−1)...(n−(S−n−1)) in Frage kommen und alle Werte genau 2-mal vorkommen. D.h.
Gewichtung =(n über S−n).
Prüfung: $\prod_{i=0,...,(S-n-2)}[(S-n-(S-n-1))+i]$ ∗(n über S−n)/(S−n−1)!
= (S−n−1)! ∗(n über S−n)/(S−n−1)! = (n über S−n).

$S_D=S-n+n=S$ [n<S/2]:
Genau n Werte werden wiederholt, die alle in Frage kommen, wovon der 1. $x_1=2$ bis $x_1=S-2n+2$ (=S−n+n−(n−1)∗2) Mal wiederholt wird, der 2. $x_2=2$ bis $x_2=S-2n+4-x_1$ (=S−n+n−(n−2)∗2−x_1) Mal, ... , der (n−1). $x_{n-1}=2$ bis $x_{n-1}=S-2-\sum_{i=1,...,(n-2)}[x_i]$ (=S−n+n−1∗2−$\sum_{i=1,...,(n-2)}[x_i]$) Mal, der n. entsprechend $x_n=S-\sum_{i=1,...,(n-1)}[x_i]$.
Gewichtung =(S−n−1 über n−1).

(2) $\sum_{j=1,...,(S-n)}$[(S−n−1 über j−1)∗(n über j)] ⇒ nach Knuth,Bd.I,S.53,(7)
= $\sum_{j=1,...,(S-n)}$[j/(S−n) ∗(S−n über j)∗(n über j)]
= 1/(S−n)∗$\sum_{j=1,...,(S-n)}$[j∗(S−n über j)∗(n über j)] ⇒ nach Knuth,Bd.I,S.59,(Prob.1)
= 1/(S−n)∗(S−n+n−1 über n−1)∗(S−n)
= (S−1 über n−1)

(2a) $\sum_{k=1,...,(S-n)}[(S-n+k)*(S-n-1 \text{ über } k-1)*(n \text{ über } k)]/Norm$
$= S * \sum_{k=1,...,(S-n)}[(S-n-1 \text{ über } k-1)*(n \text{ über } k)]/Norm$
$- n * \sum_{k=1,...,(S-n)}[(S-n-1 \text{ über } k-1)*(n \text{ über } k)]/Norm$
$+ \sum_{k=1,...,(S-n)}[k*(S-n-1 \text{ über } k-1)*(n \text{ über } k)]/Norm$
$= (S-n) + n*(S-n)/(S-1)$
$= (S+n-1)*(S-n)/(S-1)$

(2b) $\sum_{k=1,...,(S-n)}[k*(S-n-1 \text{ über } k-1)*(n \text{ über } k)]/Norm \Rightarrow$ mit: $(n \text{ über } k) = n/k*(n-1 \text{ über } k-1)$
$= n*\sum_{k=1,...,(S-n)}[(S-n-1 \text{ über } k-1)*(n-1 \text{ über } k-1)]/Norm$
$= n*(S-2 \text{ über } n-1)/(S-1 \text{ über } n-1) \Rightarrow$ mit: $(S-1 \text{ über } n-1) = (S-2 \text{ über } n-1)*(S-1)/(S-n)$
$= n*(S-n)/(S-1)$

(3) $\sum_{S-n<s\le 2*(S-n)}[s(S-n)/S^2*Gw(s)]/Norm$
$= (S-n)/S^2 * \sum_{S-n<s\le 2*(S-n)}[s*Gw(s)]/Norm$
$= (S-n)/S^2 * \sum_{k=1,...,(S-n)}[(S-n+k)*(S-n-1 \text{ über } k-1)*(n \text{ über } k)]/Norm \Rightarrow$ (2a)
$= (S+n-1)*(S-n)^2/(S^2*(S-1))$

(4) $\sum_{S-n<s\le 2*(S-n)}[(n-1)(S-s)/S^2*Gw(s)]/Norm$
$= (n-1)/S^2 * \sum_{S-n<s\le 2*(S-n)}[(S-s)*Gw(s)]/Norm \Rightarrow$ mit: $S-s=S-S+n-k$
$= (n-1)/S^2 * \sum_{k=1,...,(S-n)}[(n-k)*(S-n-1 \text{ über } k-1)*(n \text{ über } k)]/Norm$
$= n*(n-1)/S^2 * \sum_{k=1,...,(S-n)}[(S-n-1 \text{ über } k-1)*(n \text{ über } k)]/Norm$
$-(n-1)/S^2 * \sum_{k=1,...,(S-n)}[k*(S-n-1 \text{ über } k-1)*(n \text{ über } k)]/Norm \Rightarrow$ (2) und (2b)
$= n*(n-1)/S^2 - (n-1)/S^2*n*(S-n)/(S-1)$
$= [n*(n-1)*(S-1) - n*(n-1)*(S-n)]/(S^2*(S-1))$
$= n*(n-1)^2/(S^2*(S-1))$

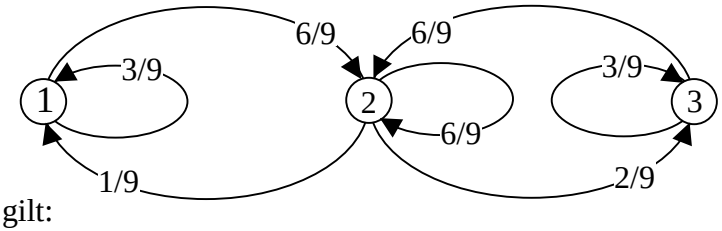
(5) $\sum_{S-n<s\le 2*(S-n)}[(1-(n-1)/S+s*(2n-S-1)/S^2)*Gw(s)]/Norm$
$= \sum_{k=1,...,(S-n)}[(1-(n-1)/S+(S-n+k)*(2n-S-1)/S^2)*(S-n-1 \text{ über } k-1)*(n \text{ über } k)]/Norm \Rightarrow$
mit: $S^2 - S*(n-1) + (S-n+k)*(2n-S-1)$
$= S^2 - S*n + S + S*(2n-S-1) - n*(2n-S-1) + k*(2n-S-1)$
$= S^2 - S*n + S + 2S*n - S^2 - S - 2n^2 + S*n + n + k*(2n-S-1)$
$= 2S*n - 2n^2 + n + k*(2n-S-1)$
$= 2n*(S-n+1/2) + k*(2n-S-1)$
$= 2n*(S-n+1/2)/S^2 * \sum_{k=1,...,(S-n)}[(S-n-1 \text{ über } k-1)*(n \text{ über } k)]/Norm$
$+(2n-S-1)/S^2 * \sum_{k=1,...,(S-n)}[k*(S-n-1 \text{ über } k-1)*(n \text{ über } k)]/Norm \Rightarrow$ (2) und (2b)
$= [2n*(S-n+1/2) + (2n-S-1)*n*(S-n)/(S-1)]/S^2$
$= [2n*(S-n+1/2) + (2(n-1)-(S-1))*n*(S-n)/(S-1)]/S^2$
$= [2n*(S-n+1/2) + 2n*(n-1)*(S-n)/(S-1) - n*(S-n)]/S^2$
$= [n*(S-n+1)*(S-1) + 2n*(n-1)*(S-n)]/(S^2*(S-1))$

(6) $P(W_{neu}=W_{alt}+1 \text{ für } S>n\geq S/2) + P(W_{neu}=W_{alt}-1 \text{ für } S>n\geq S/2) + P(W_{neu}=W_{alt} \text{ für } S>n\geq S/2)$
$= (S-n)^2*(S+n-1)/[S^2*(S-1)] + n*(n-1)^2/[S^2*(S-1)] + [n*(S-n+1)*(S-1) +2n*(n-1)*(S-n)] /[S^2*(S-1)]$
$= [(S-n)^2*(S+n-1) + n*(n-1)^2 + n*(S-n+1)*(S-1) + 2n*(n-1)*(S-n)] /[S^2*(S-1)]$
$= [(S^2-2nS+n^2)*(S+n-1) + (n^2-2n+1)*n +(S^2-nS+S-S+n-1)*n + 2n*(nS-n^2-S+n)] /[S^2*(S-1)]$
$= [S^3-2nS^2+n^2S+nS^2-2n^2S+n^3-S^2+2nS-n^2+n^3-2n^2+n+nS^2-n^2S +n^2-n +2n^2S-2n^3-2nS+2n^2] /[S^2*(S-1)]$
$= [S^3-S^2+n*(-2S^2+S^2+2S+1+S^2-1-2S)+n^2*(S-2S-1-2-S+1+2S+2) +n^3*(1+1-2)] /[S^2*(S-1)]$
$= 1$ (**P**).

(7) $(S-n)^2*(S+n-1)/[S^2*(S-1)] = n*(n-1)^2/[S^2*(S-1)] \Leftrightarrow$
$(S-n)^2*(S+n-1) = n*(n-1)^2 \Leftrightarrow$
$0 = S^3-2nS^2+Sn^2+nS^2-2Sn^2+n^3-S^2+2nS-n^2-n^3+2n^2-n \Leftrightarrow$
$0 = S^2(S-1)-n(S-1)^2-n^2(S-1) \Leftrightarrow 0 = n^2+(S-1)n-S^2 \Rightarrow$
$n = (-S+1 +\sqrt{(S-1)^2+4*S^2}) /2 = (-S+1 +\sqrt{5S^2-2S+1})/2 \Rightarrow$
$\approx (-255 +572)/2 \Rightarrow$ $n \approx 158{,}5$ $\rightarrow 61{,}911501555\%$ für S=256,
$\approx (-65535 +146542{,}5)/2 \Rightarrow$ $n \approx 40503{,}75$ $\rightarrow 61{,}803817749\%$ für S=65536,
$\approx (-16777215 +37514995)/2 \Rightarrow n \approx 10368890 \rightarrow 61{,}803400517\%$ für S=16777216.

(8) S=2: Mit
$P(W_{neu}=W_{alt}+1) = (n+1)*(2-n)^2/4 \Rightarrow$ n=1: 1/2; n=2: 0,
$P(W_{neu}=W_{alt}-1) = n*(n-1)^2/4 \Rightarrow$ n=1: 0; n=2: 1/2,
$P(W_{neu}=W_{alt}) = n*(2-n+1)/4 \Rightarrow$ n=1: 1/2; n=2: 1/2
gilt:
$P(W=1) = P(W=1)*P(W_{neu}=W_{alt},W=1) + P(W=2)*P(W_{neu}=W_{alt}-1,W=2)$
$P(W=2) = P(W=1)*P(W_{neu}=W_{alt},W=2) + P(W=2)*P(W_{neu}=W_{alt}+1,W=1)$
$P(W=1)+P(W=2) = 1$
$\Rightarrow P(W=1) = P(W=1)*1/2 + P(W=2)*1/2,$
$P(W=2) = P(W=1)*1/2 + P(W=2)*1/2,$
$P(W=1)+P(W=2) = 1$
$\Rightarrow P(W=1) = P(W=2), 2*P(W=1) = 1$
$\Rightarrow P(W=1) = 1/2$ und $P(W=2) = 1/2$.

(9) S=3: Mit
$P(W_{neu}=W_{alt}+1) = (2+n)*(3-n)^2/18 \Rightarrow$ n=1: 6/9; n=2: 2/9; n=3: 0,
$P(W_{neu}=W_{alt}-1) = n*(n-1)^2/18 \Rightarrow$ n=1: 0; n=2: 1/9; n=3: 6/9,
$P(W_{neu}=W_{alt}) = [4-n +(n-1)*(3-n)]*n/9 \Rightarrow$ n=1: 3/9; n=2: 6/9; n=3: 3/9



gilt:
$P(W=1) = P(W=1)*P(W_{neu}=W_{alt},W=1) + P(W=2)*P(W_{neu}=W_{alt}-1,W=2),$
$P(W=2) = P(W=2)*P(W_{neu}=W_{alt},W=2) + P(W=1)*P(W_{neu}=W_{alt}+1,W=1)$
$+ P(W=3)*P(W_{neu}=W_{alt}-1,W=3),$
$P(W=3) = P(W=3)*P(W_{neu}=W_{alt},W=3) + P(W=2)*P(W_{neu}=W_{alt}+1,W=2),$
$P(W=1)+P(W=2)+P(W=3) = 1$
$\Rightarrow P(W=1) = P(W=1)*3/9 + P(W=2)*1/9,$
$P(W=2) = P(W=2)*6/9 + P(W=1)*6/9 + P(W=3)*6/9,$
$P(W=3) = P(W=3)*3/9 + P(W=2)*2/9,$

P(W=1)+P(W=2)+P(W=3) = 1
$\Rightarrow$ 9∗P(W=1) = 3∗P(W=1) + P(W=2),
9∗P(W=2) = 6∗P(W=2) + 6∗P(W=1) + 6∗P(W=3),
9∗P(W=3) = 3∗P(W=3) + 2∗P(W=2),
P(W=1)+P(W=2)+P(W=3) = 1
$\Rightarrow$ P(W=1) = P(W=2)/6,
P(W=2) = 2∗P(W=1) + 2∗P(W=3),
P(W=3) = P(W=2)/3,
P(W=2)+6∗P(W=2)+2∗P(W=2) = 6
$\Rightarrow$ P(W=2) = 6/9, P(W=1) = 1/9 und P(W=3) = 2/9
$\Rightarrow$ N = 2 mit P(W=N) = 6/9.

(10) Es ergibt sich mit den Determinanten

$$D := \begin{vmatrix} 1 & 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 1 \\ p_1 & a_2 & m_3 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & p_2 & a_3 & m_4 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & p_3 & a_4 & m_5 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & p_4 & a_5 & \dots & 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots & p_{n-3} & a_{n-2} & m_{n-1} & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & p_{n-2} & a_{n-1} & m_n \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & p_{n-1} & a_n \end{vmatrix}, \quad b := \begin{vmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ \dots \\ 0 \\ 0 \\ 0 \end{vmatrix}$$

und $\mathbf{D_i}$:= {Spalte i in D durch b ersetzt} für alle i=1,...,S
P(W=i) = D_i/D als Lösung des Gleichungssystems.

[Da der Faktor f := $1/(S^2*(S-1))^{(n-1)}$ sowohl vor D als auch vor die D_i (jeweils n−1 entsprechende Zeilen) gezogen werden kann, würden sich die Werte von D_i/D nicht ändern, falls statt der bisherigen a_n, p_n und m_n folgende Werte verwendet werden:
$a_n = n*(S-n+1)*(S-1) + 2n*(n-1)*(S-n) - S^2*(S-1)$,
$p_n = (S+n-1)*(S-n)^2$,
$m_n = n*(n-1)^2$.]

Es gilt weiter:

$$D = \begin{vmatrix} a_2 & m_3 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ p_2 & a_3 & m_4 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & p_3 & a_4 & m_5 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & p_4 & a_5 & \dots & 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & p_{n-3} & a_{n-2} & m_{n-1} & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & p_{n-2} & a_{n-1} & m_n \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & p_{n-1} & a_n \end{vmatrix} - p_1 * \begin{vmatrix} a_3 & m_4 & 0 & \dots & 0 & 0 & 0 & 0 \\ p_3 & a_4 & m_5 & \dots & 0 & 0 & 0 & 0 \\ 0 & p_4 & a_5 & \dots & 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & p_{n-3} & a_{n-2} & m_{n-1} & 0 \\ 0 & 0 & 0 & \dots & 0 & p_{n-2} & a_{n-1} & m_n \\ 0 & 0 & 0 & \dots & 0 & 0 & p_{n-1} & a_n \end{vmatrix}$$

$$+ p_1*p_2* \begin{vmatrix} a_4 & m_5 & \dots & 0 & 0 & 0 & 0 \\ p_4 & a_5 & \dots & 0 & 0 & 0 & 0 \\ \dots & \dots & & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & p_{n-3} & a_{n-2} & m_{n-1} & 0 \\ 0 & 0 & \dots & 0 & p_{n-2} & a_{n-1} & m_n \\ 0 & 0 & \dots & 0 & 0 & p_{n-1} & a_n \end{vmatrix} - p_1*p_2*p_3* \begin{vmatrix} a_5 & m_6 & \dots & 0 & 0 & 0 & 0 \\ \dots & \dots & & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & p_{n-3} & a_{n-2} & m_{n-1} & 0 \\ 0 & 0 & \dots & 0 & p_{n-2} & a_{n-1} & m_n \\ 0 & 0 & \dots & 0 & 0 & p_{n-1} & a_n \end{vmatrix}$$

...

$$+/-p_1*\dots*p_{n-3}* \begin{vmatrix} a_{n-1} & m_n \\ p_{n-1} & a_n \end{vmatrix} -/+p_1*\dots*p_{n-3}*p_{n-2}*a_n \quad +/-p_1*\dots*p_{n-3}*p_{n-2}*p_{n-1}.$$

Sei

$$\mathbf{A_k} := \begin{vmatrix} a_k & m_{k+1} & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ p_k & a_{k+1} & m_{k+2} & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & p_{k+1} & a_{k+2} & m_{k+3} & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & p_{k+2} & a_{k+3} & \dots & 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & p_{n-3} & a_{n-2} & m_{n-1} & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & p_{n-2} & a_{n-1} & m_n \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & p_{n-1} & a_n \end{vmatrix}$$ für k=2,...,n−1,

$\mathbf{A_n} := a_n$, $\mathbf{A_{n+1}} := 1$.

Dann gilt für k=2,...,n−1:

$A_k = a_k*A_{k+1} - p_k*m_{k+1}*A_{k+2}. \Rightarrow$

$D = A_2 - p_1*A_3 + p_1*p_2*A_4 + /-...+/-p_1*...*p_{n-3}*A_{n-1} -/+ p_1*...*p_{n-2}*A_n -/+ p_1*...*p_{n-2}*p_{n-1}$

$\Rightarrow$ mit: $\mathbf{q_{2,2}}:= 1$, $\mathbf{q_{2,k}}:= (-1)^k * \prod_{h=1,...,k-2}[p_h]$ für alle k=3,...,n+1

$= \sum_{k=2,...,n+1}[q_{2,k}*A_k] \Rightarrow$ mit: $\mathbf{q_{3,3}}:= q_{2,3}+q_{2,2}*a_2$, $\mathbf{q_{3,4}}:= q_{2,4}-q_{2,2}*p_2*m_3$, $\mathbf{q_{3,k}}:= q_{2,k}$, k=5,...,n+1

$D = \sum_{k=3,...,n+1}[q_{3,k}*A_k] \Rightarrow$ allgemein mit j=3,...,n:

$D = \sum_{k=j,...,n+1}[q_{j,k}*A_k]$, $\mathbf{q_{j,j}}:= q_{j-1,j} + q_{j-1,j-1}*a_{j-1}$, $\mathbf{q_{j,j+1}}:= q_{j-1,j+1} - q_{j-1,j-1}*p_{j-1}*m_j$,
$\mathbf{q_{j,k}}:= q_{j-1,k}$, k=j+2,...,n+1.

Für die D_i, i=1,...,n gilt entsprechend:

$D_i = q_{i+1,i+1}*A_{i+1}$, **i=1:** $\mathbf{q_{2,2}}:= 1$, **i>1:** $\mathbf{q_{i+1,i+1}}:= (-1)^{(i+1)} * \prod_{h=1,...,i-1}[p_h]$, $\mathbf{q_{i+1,i+2}}:= 0$.

$\Rightarrow$ allgemein mit j=i+2,...,n und i<n:

$D_i = q_{j,j}*A_j + q_{j,j+1}*A_{j+1}$, $\mathbf{q_{j,j}}:= q_{j-1,j} + q_{j-1,j-1}*a_{j-1}$, $\mathbf{q_{j,j+1}}:= -q_{j-1,j-1}*p_{j-1}*m_j$.

Die hier für D und D_i gewählte Darstellung ermöglicht eine relativ einfache Berechnung dieser Werte durch ein entsprechendes Programm.

(11) Gegeben seien die Verhältnisse von (9) oben. Dann gilt:

$$B = \begin{pmatrix} P_{1,1} & P_{2,1} & 0 \\ P_{1,2} & P_{2,2} & 0 \\ 0 & P_{2,3} & 0 \end{pmatrix} = \begin{pmatrix} 3/9 & 1/9 & 0 \\ 6/9 & 6/9 & 0 \\ 0 & 2/9 & 1 \end{pmatrix}, \quad B^2 = \begin{pmatrix} 15/81 & 9/81 & 0 \\ 54/81 & 42/81 & 0 \\ 12/81 & 30/81 & 1 \end{pmatrix}, \quad B^4 = \begin{pmatrix} 711/6561 & 513/6561 & 0 \\ 3078/6561 & 2250/6561 & 0 \\ 2772/6561 & 3798/6561 & 1 \end{pmatrix},$$

und mit N_0=1

$$B^2 * \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 15/81 \\ 54/81 \\ 12/81 \end{pmatrix}, \quad B^4 * \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 711/6561 \\ 3078/6561 \\ 2772/6561 \end{pmatrix},$$

$\Rightarrow P_3(2)=12/81 \approx 0{,}1481$
$P_3(4)=2772/6561 \approx 0{,}4225$
$P_3(8)=(711*2772+3078*3798+2772*6561)/(6561*6561) \approx 0{,}7398.$

(12) Für j=1,...,N−1 und i=1,...,N gilt

$b_{i,j}(2) = m_j * m_{j-1}$, falls j=i+2,
$= p_j * p_{j+1}$, falls i=j+2,
$= m_j * [á_j+á_{j-1}]$, falls j=i+1,
$= p_j * [á_j+á_{j+1}]$, falls i=j+1,
$= m_j*p_{j-1} + p_j*m_{j+1} + á_j^2$, falls i=j,

$b_{i,j}(3) = \sum_{k=1,...,N}[b_{i,k}(2)*b_{k,j}(1)] = b_{i,j-1}(2)*b_{j-1,j}(1) + b_{i,j}(2)*b_{j,j}(1) + b_{i,j+1}(2)*b_{j+1,j}(1)$
$= b_{i,j-1}(2)*m_j + b_{i,j}(2)*á_j + b_{i,j+1}(2)*p_j$
$= m_j * m_{j-1} * m_{j-2}$, falls j=i+3
$= p_j * p_{j+1} * p_{j+2}$, falls i=j+3
$= m_j * m_{j-1} * [á_j+á_{j-1}+á_{j-2}]$, falls j=i+2
$= p_j * p_{j+1} * [á_j+á_{j+1}+á_{j+2}]$, falls i=j+2
$= m_j * (\sum_{k=j-2,...,j}[p_k*m_{k+1}] + \sum_{k=j-1,...,j}[á_k * \sum_{t=k,...,j}[á_t]])$ falls j=i+1
$= p_j * (\sum_{k=j,...,j+2}[m_k*p_{k-1}] + \sum_{k=j,...,j+1}[á_k * \sum_{t=k,...,j+1}[á_t]])$ falls i=j+1
$= \sum_{k=j-1,...,j}[p_k*m_{k+1} * \sum_{t=k,...,k+1}[z*á_t]] + á_j^3$, falls i=j

mit $\mathbf{z}:= 2+(2k+1)*(j-t)+2j*(t-j)$ [=(2+t−j)*(j−k)+(2−t+j)*(k−j+1)]

$$b_{i,j}(4) = \sum_{k=1,...,N}[b_{i,k}(3)*b_{k,j}(1)] = b_{i,j-1}(3)*b_{j-1,j}(1) + b_{i,j}(3)*b_{j,j}(1) + b_{i,j+1}(3)*b_{j+1,j}(1)$$
$$= b_{i,j-1}(3)*m_j + b_{i,j}(3)*á_j + b_{i,j+1}(3)*p_j$$
$$= \prod_{k=j-3,...,j}[m_k], \quad \text{falls } j=i+4$$
$$= \prod_{k=j,...,j+3}[p_k], \quad \text{falls } i=j+4$$
$$= \prod_{k=j-2,...,j}[m_k] * \sum_{k=j-3,...,j}[á_k], \quad \text{falls } j=i+3$$
$$= \prod_{k=j,...,j+2}[p_k] * \sum_{k=j,...,j+3}[á_k], \quad \text{falls } i=j+3$$
$$= \prod_{k=j-1,...,j}[m_k] * (\sum_{k=j-3,...,j}[p_k*m_{k+1}] + \sum_{k=j-2,...,j}[á_k * \sum_{t=k,...,j}[á_t]]) \quad \text{falls } j=i+2$$
$$= \prod_{k=j,...,j+1}[p_k] * (\sum_{k=j,...,j+3}[m_k*p_{k-1}] + \sum_{k=j,...,j+2}[á_k * \sum_{t=k,...,j+2}[á_t]]) \quad \text{falls } i=j+2$$
$$= m_j*(\sum_{k=j-1,...,j+1}[m_k*p_{k-1}*\sum_{t=j-1,...,k}[á_t]] + \sum_{k=j-1,...,j+1}[m_k*p_{k-1}*\sum_{t=k-1,...,j}[á_t]] + \sum_{k=j-1,...,j}[á_k * \sum_{t=k,...,j}[á_t * \sum_{r=1,...,j}[á_r]]]), \quad \text{falls } j=i+1$$
$$\sum_{k=0,...,3}[á_{j-1}{}^k * á_j{}^{(3-k)}]$$
$$= p_j*(\sum_{k=j-1,...,j+1}[p_k*m_{k+1}*\sum_{t=k,...,j+1}[á_t]] + \sum_{k=j-1,...,j+1}[p_k*m_{k+1}*\sum_{t=j,...,k+1}[á_t]] + \sum_{k=j,...,j+1}[á_k * \sum_{t=k,...,j}[á_t * \sum_{r=1,...,j}[á_r]]]), \quad \text{falls } i=j+1$$
$$= \sum_{k=j-2,...,j}[p_k*m_{k+1}*\sum_{t=j-1,...,k+1}[p_t*m_{t+1}]] + \sum_{k=j-1,...,j}[p_k*m_{k+1}*\sum_{r=k,...,k+1}[á_r * \sum_{t=r,...,k+1}[z*á_t]]] + á_j{}^4, \quad \text{falls } i=j$$

mit **z**:= 3+(2k+1)*(2j−r−t)+2j*(r+t−2j) [=(3+r+t−2j)*(j−k)+(3−r−t+2j)*(k−j+1)]

$$b_{i,i+n}(n) = \prod_{k=i+1,...,i+n}[m_k] \quad \text{für alle } i+n<N, \Rightarrow \text{mit: } q := [S^2*(S-1)]^n$$
$$= (i+1)*i^2 * (i+2)*(i+1)^2 * (i+3)*(i+2)^2 * ... * (i+n)*(i+n-1)^2 /q$$
$$= (i+n)!/i!*(i+n-1)!/(i-1)!*(i+n-1)!/(i-1)!/q$$
$$= (n+i \text{ über } i)*[n!*(n+i-1 \text{ über } i-1)]^2*n!/q$$
$$= (n+i \text{ über } i)*[(n+i-1 \text{ über } i-1)]^2*(n!)^3/q$$
$$= (n+i \text{ über } i)*[(i/(n+i))*(n+i \text{ über } i)]^2*(n!)^3/q$$
$$= (i/(n+i))^2 *[(n+i \text{ über } i)*n!]^3/q$$
$$= (i/(n+i))^2 *[(i+n)!/i!]^3 / [S^2*(S-1)]^n$$
$$b_{i+n,i}(n) = \prod_{k=i,...,i+n-1}[p_k] \text{ für alle } i+n\le N,$$
$$=(S+i-1)*(S-i)^2*(S+i)*(S-(i+1))^2*(S+i+1)*(S-(i+2))^2*...*(S+i+n-2)*(S-(i+n-1))^2/[S^2*(S-1)]^n$$
$$= (S+i+n-2)!/(S+i-2)!*(S-i)!/(S-i-n)!*(S-i)!/(S-i-n)!/q$$
$$= (S+i+n-2 \text{ über } S+i-2)*n!*[(S-i \text{ über } S-i-n)*n!]^2/q$$
$$= (S+i+n-2 \text{ über } n)*(S-i \text{ über } n)^2*(n!)^3/[S^2*(S-1)]^n$$
$$b_{i,j}(n) = \prod_{k=i+1,...,j}[m_k]*\sum_{k=i,...,j}[á_k], \quad \text{falls } j=i+n-1 \text{ für alle } 1<n<N-i-1,$$
$$= \prod_{k=j,...,i-1}[p_k]*\sum_{k=j,...,i}[á_k], \quad \text{falls } i=j+n-1 \text{ für alle } 1<n\le N-j+1,$$
$$= \prod_{k=i+1,...,j}[m_k]*(\sum_{k=i-1,...,j}[p_k*m_{k+1}] + \sum_{k=i,...,j}[á_k * \sum_{t=k,...,j}[á_t]]), \quad \text{falls } j=i+n-2 \text{ für alle } 1<n<N-i+2,$$
$$= \prod_{k=j,...,i-1}[p_k]*(\sum_{k=j,...,i+1}[m_k*p_{k-1}] + \sum_{k=j,...,i}[á_k*\sum_{t=k,...,i}[á_t]]), \quad \text{falls } i=j+n-2 \text{ für alle } 1<n\le N-j+2.$$

(13) Für k=1 gilt $b_{N,t}(1)<b_{N,t+1}(1)$ für alle t=1,...,N−1, $b_{N,t+1}(1)\neq 0$. Sei die Behauptung $b_{N,t}(k)<b_{N,t+1}(k)$ für alle t=1,...,N−1, $b_{N,t+1}(1)\neq 0$ und für ein k≥1 richtig. Dann gilt für k+1:

$b_{N,t}(k+1) = b_{N,t-1}(k)*m_t + b_{N,t}(k)*á_t + b_{N,t+1}(k)*p_t$ und

$b_{N,t+1}(k+1) = b_{N,t}(k)*m_t + b_{N,t+1}(k)*á_t + b_{N,t+2}(k)*p_t$

für alle t=1,...,N−1. Sei $b_{N,t+1}(k+1)\neq 0$. Dann gilt:

$b_{N,t+1}(k+1) - b_{N,t}(k+1)$
$= b_{N,t}(k)*m_t + b_{N,t+1}(k)*á_t + b_{N,t+2}(k)*p_t - b_{N,t-1}(k)*m_t - b_{N,t}(k)*á_t - b_{N,t+1}(k)*p_t$
$= m_t*[b_{N,t}(k)-b_{N,t-1}(k)] + á_t*[b_{N,t+1}(k)-b_{N,t}(k)] + p_t*[b_{N,t+2}(k)-b_{N,t+1}(k)] \ge 0,$

da sowohl die m_t, $á_t$ und p_t größer Null als auch $b_{N,t}(k)-b_{N,t-1}(k)\ge 0$, $b_{N,t+1}(k)-b_{N,t}(k)\ge 0$ und $b_{N,t+2}(k)-b_{N,t+1}(k)\ge 0$ nach Induktions-Voraussetzung, gilt die Aussage für alle k.

(14) Herleitung für S=256:
Der **Durchschnitt der nicht vorhandenen Zeichen** d bei 5000 Iterationen ergibt sich wie folgt:
(1) 99,86 =499302/5000 > δ (542. Iteration),
(2) 94,34 =420672/(5000−541) > δ (705. Iteration) ohne Einschwingwerte (<542. Iteration),
(3) 94,10 =404253/(5000−704) > δ (705. Iteration) ohne Einschwingwerte (<705. Iteration)
⇒ δ=94 (705. Iteration)
⇒ N=162 → 162/256 = 0,6328125 ≈ 63,28 %.
⇒ 705/162 ≈ 4,35-fache durchschnittliche Überladung der N Werte für mZH beliebig
⇒ >1410/x Überladungen einer x Bytes langen Direktdatei (N_0 „klein“)
(4) mZH=3 (mit der 1571. Iteration)
⇒ 1571/162 ≈ 9,70-fache durchschnittliche Überladung der N Werte für mZH=3
⇒ >3142/x Überladungen einer x Bytes langen Direktdatei (N_0 „klein“)

(15) Herleitung für S=65536:
Der **Durchschnitt der nicht vorhandenen Zeichen-Gruppen** d bei 300000 bis 800000 Iterationen ergibt sich wie folgt:
(1) 24114,16 =12057106388/500001 > δ (359023. Iteration),
(2) 24101,33 =10628156544/(800000−359022) > δ (360363. Iteration) ohne Einschwingwerte (<359023. Iteration),
(3) 24101,28 =10595837949/(800000−360362) > δ (360363. Iteration) ohne Einschwingwerte (<360363. Iteration)
⇒ δ=24101 (360363. Iteration)
⇒ N=41435 → 41435/65536 = 0,6322479248 ≈ 63,22 %.
⇒ 360363/41435 ≈ 8,697-fache durchschnittl. Überladung der N Werte für mZH beliebig
⇒ >1441452/x Überladungen einer x Bytes langen Direktdatei (N_0 „klein“)
(4) mZH=7 (mit der 632086. Iteration)
⇒ 632086/41435 ≈ 15,25488-fache durchschnittliche Überladung der N Werte für mZH=7
⇒ >2528344/x Überladungen einer x Bytes langen Direktdatei (N_0 „klein“)

(16) Herleitung für S=16777216:
Der **Durchschnitt der nicht vorhandenen Zeichen-Gruppen** d bei 100000 bis 250000 Iterationen á 1066 Überladungen ergibt sich wie folgt:
(1) 6173447,99 =926023371853/150001 > δ (131212. Iteration),
(2) 6172513,15 =733226664399/(250000−131211) > δ (133646. Iteration) ohne Einschwingwerte (<131212. Iteration),
(3) 6172493,29 =718200456584/(250000−133645) > δ (133660. Iteration) ohne Einschwingwerte (<133646. Iteration)
(4) 6172493,28 =718114040841/(250000−133659) > δ (133660. Iteration) ohne Einschwingwerte (<133660. Iteration)
⇒ δ=6172493 (133660. Iteration)
⇒ N=10604723 → 10604723/16777216 = 0,63209 ≈ 63,21 %.
⇒ 133660∗1066/10604723 ≈ 13,43567-fache durchschnittliche Überladung der N Werte für mZH beliebig
⇒ >854889360/x Überladungen einer x Bytes langen Direktdatei (N_0 „klein“)
Beispiel: 854889360 / 5935104 = 144 Überladungen (5,66 Megabyte-Datei)
(5) mZH=9 (mit der 220406. Iteration)
⇒ 220406∗1066/10604723 ≈ 22,155486-fache durchschnittliche Überladung der N Werte für mZH=9
⇒ >1409716776/x Überladungen einer x Bytes langen Direktdatei (N_0 „klein“)
Beispiel: 1409716776 / 5935104 = 237,5 Überladungen (5,66 Megabyte-Datei)

Beschreibung des Algorithmus

Im folgenden gilt: Erweitert 1.1.1 beinhaltet Erweitert 1.1 beinhaltet Erweitert 1, Erweitert E ist davon unabhängig. Der Ausgangs-Algorithmus wird als Standard bezeichnet.. Es gilt t≥1.

Programm-Initialisierung:

1. Der Urschlüssel wird so oft wiederholt, bis 256**t t-Byte-Gruppen (d.h. t*256**t Bytes) überschritten und gekürzt oder erreicht sind, sie bilden den Ausgangsschlüssel.
2. Danach können ein oder mehrere dieser Ausgangsschlüssel- t-Byte-Gruppen angesprochen und durch jeweils angegebene Gruppeninhalte ersetzt (**überlagert/überladen**) werden.
   Es werden jeweils 2*t benachbarte Bytes betrachtet, wobei die ersten t Bytes als Repräsentanten einer Integer-Zahl mit Wertebereich von 0 to 256**t –1 aufgefasst werden, die, um 1 erhöht, die Nummer der angesprochenen Ausgangsschlüssel- t-Byte-Gruppe angibt, die mit den folgenden t Bytes, die den neuen t-Byte-Gruppen-Inhalt darstellen, überladen werden sollen.
   Allgemein können Schlüssel-Stellen auch mehrfach überladen werden. Nach einer tatsächlich Überladung ist der Überladewert der neue Schlüsselwert an der Überladestelle für die weitere Verarbeitung. Die letzte auf eine Stelle anzuwendende Überladeinformation entscheidet, welcher Wert der endgültige Schlüsselwert an dieser Stelle ist.
   Bei der Überladung wird der eingegebene bzw. aus einer Datei stammende Wert zuvor gemäß
   Überladewert := (Eingabe-/Dateiwert + bisheriger Schlüsselwert ***** Primzahl
   + Schlüsselwert an vorheriger Stelle bzw. Stelle 256**t für Stelle 1) modulo 256**t
   mit den aktuellen **Primzahlen** :=113 | 30781 | 100000000019 für t=1|2|3,
   transformiert, bevor der so abgeleitete Wert als eigentlicher Überladewert verwendet wird. Des weiteren können zu Überladeinformationen auf angegebene Stellen- und Werte-Intervalle eingeschränkt werden, um bei Verwendung "beliebiger" Dateien plumpe Überladungen zu verhindern. Nicht in den angegebenen Intervallgrenzen liegende Überladeinformationen werden ignoriert.

   Allgemein ist zu beachten, dass für alle Stellen innerhalb des Stellen-Intervalls **im Zusammenhang mit einer beliebig variierenden, (fast) beliebig langen Überladeinformation** gilt::
   Schlüsselwerte aus dem Schlüsselwerte-Intervall werden am Ende (fast) vollständig in Überladewerte des Überladewerte-Intervalls überführt.
   Falls es Überladewerte gibt, die nicht im Schlüsselwerte-Intervall liegen, gilt **noch einschränkender**:
   Alle Schlüsselwerte werden am Ende (fast) vollständig in Werte des Überladewerte-Intervalls, die nicht im Schlüsselwerte-Intervall vorkommen, überführt.
   Je besser eine Überladeinformation die genannten Bedingungen erfüllt, desto mehr gelten genannte Folgerungen, was unbedingt zu beachten ist, um Trivial-Schlüssel zu vermeiden.
   Ist ein Ausgangsschlüssel, eine Überladeinformation und ein Überladewerte-Intervall gegeben, so **gilt allgemein**:
   Schlüsselwerte-Intervalle, die dieses Überladewerte-Intervall umfassen, liefern den gleichen Ergebnisschlüssel, falls für alle Schlüsselwerte, die im Schlüsselwerte-Intervall aber nicht im Überladewerte-Intervall enthalten sind, gilt, dass sie nicht im Ausgangsschlüssel vorkommen.

   Wählt man eine beliebige Menge von Dateien, die fortlaufend chiffriert wird, so kann diese mehrfach zur Überladung verwendet werden, um einen Schlüssel zu erzeugen, der sich statistisch nicht von einem „Zufallsschlüssel“ unterscheidet(siehe "Konvergenz von Überlade-Operationen bei Schlüsseln"). Der ggf. so geänderte Ausgangsschlüssel bildet dann den **echten Schlüssel**.

3. Mit Hilfe dieses echten Schlüssels, der zunächst gemäß Schritt 4, oder nachfolgend mit Hilfe eines Operations-Schlüssels, der zunächst gemäß den Schritte 7 und 10 (in Standard-Form) modifiziert wird, wird eine **Vorgänger-S-Tabelle** und die zugehörige **inverse S-Tabelle** für die Dechiffrierung erzeugt (siehe "Einleitung/Zusammenfassung", "Substitutions-Tabellen S").

4. Die t-Byte-Gruppe in einem Schlüssel (**pval**; siehe danach), von der 2* 2*7 Bytes im Schlüssel verwendet werden, um beide Zahlen **jedes Zufallszahlen-Prozesses** (für 52 Bit Mantissen gemäß IEEE 754; für Zahlen mit x Mantissenbytes pro Zahl ist im weiteren 7 durch x Zeichen zu ersetzen) zu initialisieren, bestimmt sich durch den t-Byte-Wert +1 an der
   - 1. (oder folgenden) Stelle der S-Tabelle für den aktuellen Operations-Schlüssels als Schlüssel,
   - 1. Stelle des echten Schlüssels unter Verwendung des echten Schlüssels als Schlüssel.

   Die Operationen im Zusammenhang mit den Zufallszahlen eines Prozesses laufen dabei wie folgt ab:
   A. Bei der **Initialisierung** der Zufallszahlen des **Prozesses x**=1,2 werden nur die wenigst signifikanten Bits des ersten Schlüssel-Bytes verwendet, um die meist signifikanten Bits der Mantisse der 1. Zufallszahl zu setzen (IEEE 754-spezifisch, für andere Gleitkomma-Darstellungen anders), analog werden die wenigst signifikanten Bits des zweiten Schlüssel-Bytes verwendet, um die meist signifikanten Bits der Mantisse der 2. Zufallszahl zu setzen.
      a. Nun werden die weiteren Schlüssel-Bytes (Bytegruppen-übergreifend) jeweils abwechselnd in die 1. und 2. Zufallszahl vom meist zum wenigst signifikanten Byte übertragen.
      b. Die Exponenten beider Zufallszahlen werden so modifiziert, dass jeder Exponent den Wert Null darstellt.
      c. Das Ergebnis der Multiplikation des Quadrats der 1. mit dem Quadrat der 2. Zufallszahl ergibt die neue 1. Zufallszahl, während das Quadrat der ursprünglichen 1. Zufallszahl als neue 2. Zufallszahl zugewiesen wird. Die Exponenten beider Zufallszahlen werden danach wieder auf Null zurückgesetzt.
      d. Ein **Vergleichswert** (**compval[x]**) wird erzeugt, um die Notwendigkeit der Änderung des laufenden Operations-Schlüssels (Standard) oder der Komponenten (Erweitert 1) zu ermitteln:
         compval[x] = ranvalpx and signbits
         signbits = Vergleichsbits, 2**(gesetzte Bits in signbits) definiert die durchschnittliche Anzahl unterschiedlicher Werte zwischen identischen compval-Werten
         ranvalpx = ranpxn1[b3]*256+ranpxn1[b4]
         | ((ranpxn1[b1]*256+ranpxn1[b2])*256+ranpxn1[b3])*256+ranpxn1[b4]
         für t=1|2 oder 3,
         ranpx**ny[bz]** = Byte z der y. Zufallszahl von Prozess x.
      e. Schritt 4.A.c wird erneut ausgeführt.
   B. Bei **Anwendung** der Zufallszahlen auf den (echten/Operations-)Schlüssel bzw. die Daten werden pro 8 Bytes (IEEE 754-spezifisch) bzw. für die restlichen Bytes des Schlüssels bzw. der Daten folgende Schritte wiederholt:
      a. Beide Zufallszahlen werden quadriert.
      b. Das jeweils wenigst signifikante Mantissenbyte (bei Datenanwendung muss die Anzahl der einbezogenen Mantissenbytes ein Vielfaches von t sein) sowie das jeweils meist signifikante Mantissen-Halbbyte jeder Zufallszahl werden zunächst nicht berücksichtigt.
      c. Die folgenden 4 (IEEE 754-spezifisch) Mantissenbytes (vom meist zum wenigst signifikanten) zunächst der 1. und danach der 2. Zufallszahl werden per ‚xor'-Operation auf anstehende Schlüssel- bzw. Datenbytes angewendet, soweit gemäß Datenlänge möglich.
      d. Das folgende (5.) Mantissenbyte der 1. Zufallszahl wird zusätzlich auf das 1. durch die 2. Zufallszahl schon behandelte Schlüssel- bzw. Datenbyte per ‚xor'-Operation angewendet, soweit möglich. Analog wird das folgende Mantissenbyte der 2. Zufallszahl mit dem 1. durch die 1. Zufallszahl schon behandelte Schlüssel- bzw. Datenbyte verknüpft.
      e. Die 1. Zufallszahl wird zur 2. Zufallszahl, die 1. Zufallszahl selbst wird durch das Produkt der alten 1. mit der alten 2. Zufallszahl ersetzt.
      f. Die Exponenten beider Zufallszahlen werden wieder auf Null zurückgesetzt.

g. Falls im Laufe der Bearbeitung der Zyklus des aktuellen Pseudo-Zufallszahlen-Prozesses bereits einmal durchlaufen sein sollte, d.h. die Mantissenbytes des Prozesses sind identisch mit denen am Anfang, dann werden folgende Aktionen veranlasst:
   i. Der zuvor verwendete **Stellenwert** (**pval**) der t-Byte-Stelle im (echten/Operations-)Schlüssel, ab der die letzten 2*7 Zeichen im Schlüssel zur Initialisierung der beiden Zahlen eines neuen Prozesses verwendet wurden, wird um (pval+7) modulo (t*256**(t–1) –7 +1) erhöht.
   ii. Es wird ein neuer Pseudo-Zufallszahlen-Prozess initialisiert (Schritt 4.A) und die so erzeugten Pseudo-Zufallszahlen werden weiter mit dem restlichen Schlüssel oder Daten verknüpft.

5. Nach Erhöhung des Stellenwertes pval auf (pval+7) modulo (t/2*256**t –7 +1) werden der **Pseudo-Zufallszahlen-Prozess 1** und analog der **Prozess 2** gemäß Schritt 4.A initialisiert und die so erzeugten Pseudo-Zufallszahlen werden mit dem aktuellen (Operations-)Schlüssel per ‘xor’-Operation verknüpft, was zu einem **modifizierten Operations-Schlüssel** (**mopkey**) führt.
6. Der modifizierte Operations-Schlüssel wird t-Byte-weise via **S-Tabelle** umgewandelt.
7. In einer Schleife mit **m** Durchläufen werden folgende Operations-Schlüssel-Änderungen durchgeführt (Standard: **m** =2; Erweitert 1: **m** =2*t +mopkey[ranval1+1] mod (2*t+1) mit
   [Standard] (1) **ranval1** = ranp2n1[b2] | ranp2n1[b2]*256+ranp2n2[b2]
   | (ranp2n1[b2]*256+ranp2n2[b2])*256+ranp1n1[b2] für t=1|2|3,
   (2) **ranval1** = ranp1n1[b2] | ranp1n1[b2]*256+ranp1n2[b2]
   | (ranp1n1[b3]*256+ranp1n1[b3])*256+ranp2n1[b3] für t=1|2|3,
   mopkey[x] = x. t-Byte-Gruppe des modifizierten Operations-Schlüssels):
   A. **'xor' von links nach rechts**: mopkey[i+1] = mopkey[i+1] xor mopkey[i], i=1,...,256**t –1.
   B. t-Byte-weise Umwandlung via **S-Tabelle**.
   C. Der **Verschiebungswert** in Bytes wird zu **svalb1** = mopkey[j]*t xor ranval2 mit [(1) Standard]
      (1) **j** = 256**t, **ranval2** = ranp1n1[b1] | ranp1n1[b1]*256+ranp1n1[b2]
      | (ranp1n1[b1]*256+ranp1n1[b2])*256+ranp1n1[b3] für t=1|2|3,
      (2) **j** = 1, **ranval2** = ranp2n1[b1] | ranp2n1[b1]*256+ranp2n2[b2]
      | (ranp2n1[b1]*256+ranp2n1[b2])*256+ranp2n1[b3] für t=1|2|3.
   D. **'xor' von rechts nach links**: mopkey[i–1] = mopkey[i–1] xor mopkey[i], i=256**t,...,2.
   E. t-Byte-weise Umwandlung via **S-Tabelle**.
   F. Der modifizierte Operations-Schlüssel wird um **svalb1** Bytes (siehe Schritt 7.C) **zyklisch verschoben** (das Ende wird als an den Anfang angrenzend angesehen).
8. Die durch den **Zufallszahlen-Prozess 2** erzeugten Zufallszahlen werden mit dem modifizierte Operations-Schlüssel per 'xor'-Operation gemäß Schritt 4.B verknüpft.
9. Der modifizierte Operations-Schlüssel wird t-Byte-weise via **S-Tabelle** umgewandelt.
10. Im Zusammenhang mit dem aktuell modifizierten Operations-Schlüssel wird **erneut** der Schritt 7 (mit (2) statt (1)) in Form der Schritte 7.D und 7.E gefolgt von Schritt 7.C (mit (2) statt (1)) gefolgt von den Schritten 7.A, 7.B und 7.F ausgeführt.

11. Nach Erzeugung einer **Vorgänger-S-Tabelle** gemäß Schritt 3 und dem Setzen von **pval=1**, wird der 1. Übergangs-Operations-Schlüssel durch den **1. Durchlauf der Schritte 4 bis 10** erzeugt.
12. Unter Verwendung des 1. Übergangs-Operations-Schlüssels werden die **eigentliche S-Tabelle** und **deren inverse S-Tabelle** erzeugt, die im Folgenden verwendet werden (siehe "Einleitung/Zusammenfassung", "Substitutions-Tabellen S").
13. Der **nächste Durchlauf der Schritte 4 bis 10** erzeugt den 2. Übergangs-Operations-Schlüssel, in dem eine t-Byte-Gruppe durch den t-Byte-Wert +1 an der 2. Stelle der S-Tabelle bestimmt wird. Die 2* 2*7 Bytes dieser (und folgender) t-Byte-Gruppe initialisieren die 4 Zahlen der **2 Zufallszahlen-Prozesse** gemäß Schritt 4.A, die verwendet werden, um die Daten-Bytes zu behandeln.
14. Ein **weiterer Durchlauf der Schritte 4 bis 10** erzeugt den **Ausgangs-Operations-Schlüssel**, der für die Behandlung der Daten Verwendung findet.

I. Datenbearbeitungs-Initialisierung:

15. Für jede separat zu behandelnde Datei bzw. Datenmenge werden die eigentliche und deren inverse S-Tabelle (Schritt 12), die Pseudo-Zufallszahlen-Prozesse zur Daten-Behandlung (Schritt 13), der Stellenwert pval zur Zufallszahlen-Initialisierung (Schritt 4.B.g.i) und der Ausgangs-Operations-Schlüssel (Schritt 14) wiederhergestellt, falls keine Stromchiffrierung stattfindet.

II. Datenbearbeitung pro Datenblock:

16. Falls der einzige Datenblock einer Datei weniger als t Bytes umfasst, dann werden die vorliegenden Datenbytes beim Chiffrieren wie beim Dechiffrieren jeweils einzeln mit den Mantissenbytes beider Zufallszahlen beider Pseudo-Zufallszahlen-Prozesse sowie mit Operationsschlüssel-Bytes per ‚xor'-Operation gemäß den Schritten 20.C, 20.G und 20.L behandelt. In diesem Fall werden keine weiteren Schritte ausgeführt.

17. Im Fall einer variablen **Datenblocklänge** in Bytes (**blklen**) kann diese Länge zwischen minimaler und doppelt minimaler als maximaler Datenblocklänge variieren. Im Fall von fester Datenblocklänge ist minimale gleich maximale Datenblocklänge. Die **minimale Datenblocklänge** in Bytes (**minblk**) muss ein Vielfaches von **imb** = 4*t (=8 für t=1; siehe auch Schritt 4.B) sein.
    A. Falls der letzte Datenblock einer Datei kleiner als minblk ist und die beiden letzten Datenblöcke zusammen kleiner als 2*minblk sind, dann wird der letzte zusammen mit dem vorletzten Datenblock als ein Datenblock behandelt.
    B. Sonst wird die Datenblocklänge **blklen** als zwischen minblk und 2*minblk liegend bestimmt:
        blklen = (minblk/imb+(mopkey[ranval3+1] mod (minblk/imb+1)))*imb mit
        ranval3 = ranp1n1[b1] xor ranp2n1[b1] | (ranp1n1[b1]*256)+ranp2n1[b1]
        | (ranp1n1[b1]*256+ranp1n2[b1])*256+ranp2n1[b1] für t=1|2|3
    C. Erreicht oder übersteigt die Längensumme der beiden letzten Datenblöcke 2*minblk und ist die Länge des letzten Datenblocks kleiner minblk, so wird die Länge des letzten Datenblocks so um ein Vielfaches von imb erhöht, dass minblk gerade erreicht oder überschritten wird. In diesem Fall wir die Länge des vorletzten Datenblocks entsprechend reduziert.

18. Falls der Zyklus des 1. aktuellen Pseudo-Zufallszahlen-Prozesses im Laufe der Bearbeitung des letzten Datenblocks bzw. am Ende dieses Datenblocks bereits einmal durchlaufen sein sollte (Standard) oder falls die Bedingung von Schritt 19 erfüllt ist (Erweitert 1), dann werden folgende Aktionen am Anfang des folgenden/aktuellen Datenblocks veranlasst:
    A. Erweitert 1.1: Der aktuelle Operations-Schlüssel wird mit Hilfe der Parameter einer Komponentenwechsel-Parameterdatei, falls vorhanden, wie Daten chiffriert.
    B. Ein Durchlauf der Schritte 11 bis 14 erzeugt einen neuen Komponentensatz (siehe "Einleitung/ Zusammenfassung", "Komponenten-Wechsel"), der ab jetzt der Daten-Behandlung dient.

19. Falls die gesetzten Bits in **currentval[1]** mit denen in **compval[1]** (s. Schritt 4.A.d; Standard) mit
    currentval[x] = ranvalpx and signbits
    (siehe Schritt 4.A.d für Zufallsprozess x und mit laufenden Zufallszahlen),
    signbits = 2**q -1, q Bits mit 0<=q<$\log_2$(256**t/s) = 8*t -$\log_2$(s), wobei s die Anzahl der Anwendungen des Operations-Schlüssels pro Daten-Block ist, d.h. s=blklen/(t*256**t)
    = 63 | 8388607 | 4294967295 zur Zeit für t=1|2|3,
    identisch sind oder die laufende Summe der blklen bzgl. der aktuellen Komponenten größer als t*256**t +1 ist (Erweitert 1), dann werden die folgenden Aktionen ausgeführt:
    A. Erweitert 1.1.1: Der aktuelle Operations-Schlüssel wird mit Hilfe der Parameter einer Komponentenwechsel-Parameterdatei, falls vorhanden, wie Daten chiffriert.

B. Falls **currentval[2]>currentval[1]**, dann werden die Schritte 7.A bis 7.F mit dem aktuellen Operations-Schlüssel in einer Schleife von **m** Durchläufen (**m** =2 konstant z.Zt.) durchgeführt.
C. Sonst falls **currentval[2]<=currentval[1]**, dann werden die Schritte 7.D und 7.E gefolgt von Schritt 7.C (mit (2) statt (1)) gefolgt von den Schritten 7.A, 7.B und 7.F in einer Schleife von **m** Durchläufen (**m** =2 konstant z.Zt.) durchgeführt.

20. Falls die Daten **chiffriert** werden sollen (prinzipiell können „chiffriert" und „dechiffriert" als Begriffe einheitlich im gesamten Text vertauscht werden), dann werden folgende Aktionen veranlasst, **mainblk** = blklen–(blklen mod t), **ranval4/5/6** zu Anfang der Datenblock-Bearbeitung erzeugt:
A. Falls blklen mod t ≠ 0 ist, dann werden das 2. und folgende Byte(s) **jeder Zufallszahl** (allgemein: t-1 Bytes der Zufallszahlen) beider Zufalls-Prozesse **gespeichert**, um damit in den Schritten 20.E.c und 20.O.c die letzten bis zu t-1 Datenblock-Byte(s) zu behandeln.
B. Die (ersten) mainblk Datenbytes werden t-Byte-weise via **S-Tabelle** umgewandelt.
C. Durch Erzeugung von Zufallszahlen des **1. Zufalls-Prozesse**s (Schritt 4.B) werden die Mantissenbytes dieser Zufallszahlen fortlaufend per 'xor'-Operation mit allen mainblk Datenbytes verknüpft.
D. Falls im Laufe der Bearbeitung der Zyklus des aktuellen 1. Pseudo-Zufallszahlen-Prozesses bereits einmal durchlaufen sein sollte, dann werden folgende Aktionen veranlasst:
   a. Der zuvor verwendete **Stellenwert** (**pval**) der t-Byte-Stelle im Operations-Schlüssel, ab der die letzten 2*7 Zeichen im Operations-Schlüssel zur Initialisierung der beiden Zahlen eines neuen Prozesses verwendet wurden, wird um (pval+7) modulo (t*256**(t–1) –7 +1) erhöht.
   b. Es wird ein neuer 1. Pseudo-Zufallszahlen-Prozess initialisiert (Schritt 4.A) und die so erzeugten Pseudo-Zufallszahlen werden weiter per 'xor'-Operation mit den restlichen der mainblk Datenbytes verknüpft (Schritt 4.B).
E. Falls blklen mod t ≠ 0 ist, dann erfolgt mit **block[x]** = x. Byte des Datenblocks:
   a. Der Datenblock wird durch Verschieben und Kopieren des Bytes vor dem letzten und weiterer Bytes davor zu einer **neuen t-Byte-Gruppe** erweitert:
      block[blklen+t–(blklen mod t)–i] = block[blklen–i], i=0,...,t–1.
   b. Diese neue t-Byte-Gruppe wird via **S-Tabelle** umgewandelt (siehe Schritt 20.B).
   c. Die in Schritt 20.A gespeicherten Bytes des **1. Zufallszahlen-Prozesses** werden per 'xor'-Operation mit der neuen t-Byte-Gruppe verknüpft (analog zu Schritt 4.B).
   d. Die zuvor zur Erweiterung dienenden Bytes werden auf ihre Ausgangspositionen **zurückkopiert**:
      block[blklen–i] = block[blklen+t–(blklen mod t)–i], i= t–1,...,0.
   e. Die Bytes mainblk+1 bis blklen werden per 'xor'-Operation **verknüpft** mit **und ersetzen** die Bytes mainblk+1–t bis blklen–t der letzten (**n**.) t-Byte-Gruppe:
      block[blklen–t–i] = block[blklen–t–i] xor block[blklen–i], i=0,...,(blklen mod t)–1.
F. Die mainblk Datenbytes werden t-Byte-weise via **S-Tabelle** umgewandelt.
G. Es werden die Bytes des **aktuellen Operations-Schlüssels** per 'xor'-Operation
   (a) unter Mehrfach-Abarbeitung des Operations-Schlüssels falls blklen ≥ t*256**t,
   (b) unter Behandlung des jeweils 1. Teils des nächsten Abschnitts (in doppelter maximaler Datenblock-Länge) des Operations-Schlüssels, falls blklen ≤ t*(256**t)/2,
   (c) unter Abarbeitung zunächst des vorderen Teils des Operations-Schlüssels, falls t*(256**t)/2 < blklen < t*256**t
   fortlaufend mit allen mainblk Datenbytes verknüpft.
H. Erneut werden die mainblk Datenbytes t-Byte-weise via **S-Tabelle** umgewandelt.
I. In einer Schleife mit **m** Durchläufen (Standard: **m** =1; Erweitert 1: **m** = mb+mf+mv mit
   mb = 0 [Basiswert], **mf** = ($\log_2$(mainblk/t)+3) div 4 [Festwert],
   mv = mopkey[ranval4+1] mod (mf+1) [variabler Wert]
   ranval4 = ranp1n1[b2] | ranp1n1[b2]*256+ranp1n2[b2]
   | (ranp1n1[b2]*256+ranp1n2[b2])*256+ranp2n1[b2] für t=1|2|3)
   mit **n** = blklen div t, werden folgende Änderungen der mainblk Datenbytes durchgeführt:

a. **'xor' von links nach rechts**: block[i+1] = block[i+1] xor block[i], i=1,...,n–1.
b. t-Byte-weise Umwandlung via t-Byte- **S-Tabelle**.
c. **'xor' von rechts nach links**: block[i–1] = block[i–1] xor block[i], i=n,...,2.
d. t-Byte-weise Umwandlung via t-Byte- **S-Tabelle**.
e. Erweitert 1: Mit dem **Verschiebungswert** in Bytes **svalb2**, der auf
   svalb2 = ((mopkey[j]*256**t)+mopkey[j+1]) mod blklen mit
   j = (ranval5+2*i) mod 256**t, j=0 <=> j=256**t für Durchlauf i=1,...,m,
   ranval5 = ranp1n1[b3] | ranp1n1[b3]*256+ranp1n1[b4]
   | (ranp1n1[b4]*256+ranp1n1[b5])*256+ranp1n1[b6] für t=1|2|3,
   gesetzt wird, werden die blklen Datenblock-Bytes um **svalb2** Bytes **zyklisch verschoben** (das Ende wird als an den Anfang des Datenblocks angrenzend angesehen).

J. Erneut werden Schritt 20.G mit dem Mehrfachen, dem 2. Abschnitts-Teil oder dem hinteren Teil des Operations-Schlüssels und Schritt 20.H ausgeführt (**aktueller Operations-Schlüssel**).
K. Falls blklen mod t ≠ 0 ist, dann:
   a. Der Datenblock wird durch Verschieben und Kopieren des Bytes vor dem letzten und weiterer Bytes davor zu einer **neuen t-Byte-Gruppe** erweitert:
      block[blklen+t–(blklen mod t)–i] = block[blklen–i], i=0,...,t–1.
   b. Diese neue t-Byte-Gruppe wird via **S-Tabelle** umgewandelt (siehe Schritt 20.E.b).
   c. Das erste Element des Teils des **Operations-Schlüssels**, der gemäß 20.G verwendet wird, wird per 'xor'-Operation verknüpft.
   d. Erneut wird die neue t-Byte-Gruppe wird via **S-Tabelle** umgewandelt (s. Schritt 20.H).
   e. Die Schritte 20.K.b bis 20.K.d werden **erneut ausgeführt**.
   f. Die zuvor zur Erweiterung dienenden Bytes werden auf ihre Ausgangspositionen **zurückkopiert**:
      block[blklen–i] = block[blklen+t–(blklen mod t)–i], i= t–1,...,0.
L. Durch Erzeugung von Zufallszahlen des **2. Zufalls-Prozesse**s (Schritt 4.B) werden die Mantissen-bytes dieser Zufallszahlen fortlaufend per 'xor'-Operation mit allen mainblk Datenbytes verknüpft.
M. Falls im Laufe der Bearbeitung der Zyklus des aktuellen 2. Pseudo-Zufallszahlen-Prozesses bereits einmal durchlaufen sein sollte, dann werden folgende Aktionen veranlasst:
   a. Der zuvor verwendete **Stellenwert** (**pval**) der t-Byte-Stelle im Operations-Schlüssel, ab der die letzten 2*7 Zeichen im Operations-Schlüssel zur Initialisierung der beiden Zahlen eines neuen Prozesses verwendet wurden, wird um (pval+7) modulo (t*256**(t–1) –7 +1) erhöht.
   b. Es wird ein neuer 2. Pseudo-Zufallszahlen-Prozess initialisiert (Schritt 4.A) und die so erzeugten Pseudo-Zufallszahlen werden weiter per 'xor'-Operation mit den restlichen der mainblk Datenbytes verknüpft (Schritt 4.B).
N. Die mainblk Datenbytes werden t-Byte-weise via **S-Tabelle** umgewandelt.
O. Falls blklen mod t ≠ 0 ist, dann:
   a. Die Bytes mainblk+1 bis blklen werden per 'xor'-Operation **verknüpft** mit **und ersetzen** die Bytes mainblk+1–t bis blklen–t der letzten (**n**.) t-Byte-Gruppe:
      block[blklen–t–i] = block[blklen–t–i] xor block[blklen–i], i=0,...,(blklen mod t)–1.
   b. Der Datenblock wird durch Verschieben und Kopieren des Bytes vor dem letzten und weiterer Bytes davor zu einer **neuen t-Byte-Gruppe** erweitert:
      block[blklen+t–(blklen mod t)–i] = block[blklen–i], i=0,...,t–1.
   c. Die in Schritt 20.A gespeicherten Bytes des **2. Zufallszahlen-Prozesses** werden per 'xor'-Operation mit der neuen t-Byte-Gruppe verknüpft (analog zu Schritt 4.B).
   d. Die neue t-Byte-Gruppe wird via **S-Tabelle** umgewandelt (siehe Schritt 20.E.b).
   e. Die zuvor zur Erweiterung dienenden Bytes werden auf ihre Ausgangspositionen **zurückkopiert**:
      block[blklen–i] = block[blklen+t–(blklen mod t)–i], i= t–1,...,0.

P. Erweitert E: Mit dem **Verschiebungswert** in Bytes **svalb3**, der auf
svalb3 = ((mopkey[j]*256**t)+mopkey[j+1]) mod blklen mit
j = ranval6 mod 256**t, j=0 <=> j=256**t,
ranval6 = ranp1n1[b1] | ranp1n1[b1]*256+ranp1n2[b1]
| (ranp1n1[b1]*256+ranp1n2[b1])*256+ranp2n1[b1] für t=1|2|3,
gesetzt wird, werden die blklen Datenblock-Bytes um **svalb3** Bytes **zyklisch verschoben** (das Ende wird als an den Anfang des Datenblocks angrenzend angesehen).

21. Falls die Daten **dechiffriert** werden sollen, dann werden folgende Aktionen veranlasst,
    A. Erweitert E: Die zyklische Verschiebung innerhalb des Datenblocks gemäß Schritt 20.P wird zurückgenommen
    B. Falls blklen mod t ≠ 0 ist, dann werden das 2. und folgende Byte(s) **jeder Zufallszahl** (allgemein: t-1 Bytes der Zufallszahlen) beider Zufalls-Prozesse **gespeichert**, um damit in den Schritten 21.C.c und 21.L.c die letzten bis zu t-1 Datenblock-Byte(s) zu behandeln.
    C. Falls blklen mod t ≠ 0 ist, dann:
        a. Der Datenblock wird durch Verschieben und Kopieren des Bytes vor dem letzten und weiterer Bytes davor zu einer **neuen t-Byte-Gruppe** erweitert:
            block[blklen+t–(blklen mod t)–i] = block[blklen–i], i=0,...,t–1.
        b. Diese neue t-Byte-Gruppe wird via **inverse S-Tabelle** rückumgewandelt (s. Schritt 21.D).
        c. Die in Schritt 21.A gespeicherten Bytes des **2. Zufallszahlen-Prozesses** werden per ‘xor’-Operation mit der neuen t-Byte-Gruppe verknüpft (analog zu Schritt 4.B).
        d. Die zuvor zur Erweiterung dienenden Bytes werden auf ihre Ausgangspositionen **zurückkopiert**:
            block[blklen–i] = block[blklen+t–(blklen mod t)–i], i= t–1,...,0.
        e. Die Bytes mainblk+1 bis blklen werden per ‘xor’-Operation **verknüpft** mit **und ersetzen** die Bytes mainblk+1–t bis blklen–t der letzten (**n**.) t-Byte-Gruppe:
            block[blklen–t–i] = block[blklen–t–i] xor block[blklen–i], i=0,...,(blklen mod t)–1.
    D. Die (ersten) mainblk Datenbytes werden t-Byte-weise via **inverse S-Tabelle** rückumgewandelt.
    E. Durch Erzeugung von Zufallszahlen des **2. Zufalls-Prozesse**s (Schritt 4.B) werden die Mantissen-bytes dieser Zufallszahlen fortlaufend per ‘xor’-Operation mit allen mainblk Datenbytes verknüpft.
    F. Falls im Laufe der Bearbeitung der Zyklus des aktuellen 2. Pseudo-Zufallszahlen-Prozesses bereits einmal durchlaufen sein sollte, dann werden folgende Aktionen veranlasst:
        a. Der zuvor verwendete **Stellenwert** (**pval**) der t-Byte-Stelle im Operations-Schlüssel, ab der die letzten 2*7 Zeichen im Operations-Schlüssel zur Initialisierung der beiden Zahlen eines neuen Prozesses verwendet wurden, wird um (pval+7) modulo (t*256**(t–1) –7 +1) erhöht.
        b. Es wird ein neuer 2. Pseudo-Zufallszahlen-Prozess initialisiert (Schritt 4.A) und die so erzeugten Pseudo-Zufallszahlen werden weiter per ‘xor’-Operation mit den restlichen der mainblk Datenbytes verknüpft (Schritt 4.B).
    G. Falls blklen mod t ≠ 0 ist, dann:
        a. Der Datenblock wird durch Verschieben und Kopieren des Bytes vor dem letzten und weiterer Bytes davor zu einer **neuen t-Byte-Gruppe** erweitert:
            block[blklen+t–(blklen mod t)–i] = block[blklen–i], i=0,...,t–1.
        b. Diese neue t-Byte-Gruppe wird via **inverse S-Tabelle** rückumgewandelt (s. Schritt 21.D).
        c. Das erste Element des Teils des **Operations-Schlüssels**, der gemäß 21.I verwendet wird, wird per ‘xor’-Operation verknüpft.
        d. Erneut wird die neue t-Byte-Gruppe via **inverse S-Tabelle** rückumgewandelt (s. Schritt 21.D).
        e. Die Schritte 21.G.b bis 21.G.d werden **erneut ausgeführt**.
        f. Die zuvor zur Erweiterung dienenden Bytes werden auf ihre Ausgangspositionen **zurückkopiert**:
            block[blklen–i] = block[blklen+t–(blklen mod t)–i], i= t–1,...,0.
    H. Die mainblk Datenbytes werden t-Byte-weise via **inverse S-Tabelle** rückumgewandelt.

I. Es werden die Bytes des **aktuellen Operations-Schlüssels** per 'xor'-Operation
(a) unter Mehrfach-Abarbeitung des Operations-Schlüssels falls blklen ≥ t*256**t,
(b) unter Behandlung des jeweils 2. Teils des nächsten Abschnitts (in doppelter maximaler Datenblock-Länge) des Operations-Schlüssels, falls blklen ≤ t*(256**t)/2,
(c) unter Abarbeitung zunächst des hinteren Teils des Operations-Schlüssels, falls t*(256**t)/2 < blklen < t*256**t
fortlaufend mit allen mainblk Datenbytes verknüpft.

J. In einer Schleife mit **m** Durchläufen (Standard: **m** =1; Erweitert 1: **m** = mb+mf+mv gemäß Schritt 20.I) werden folgende Änderungen der mainblk Datenbytes durchgeführt:
   a. Erweitert 1: Die zyklische Byte-Verschiebung innerhalb des Datenblocks gemäß Schritt 20.I wird rückgängig gemacht.
   b. t-Byte-weise Rückumwandlung via **inverse** t-Byte- **S-Tabelle**.
   c. **'xor' von rechts nach links invers**: block[i–1] = block[i–1] xor block[i], i=2,...,n.
   d. t-Byte-weise Rückumwandlung via **inverse** t-Byte- **S-Tabelle**.
   e. **'xor' von links nach rechts invers**: block[i+1] = block[i+1] xor block[i], i= n–1,...,1.

K. Erneut werden die Schritte 21.H, 21.I mit dem Mehrfachen, dem 1. Abschnitts-Teil oder dem vorderen Teil des Operations-Schlüssels und erneut 21.H ausgeführt (**akt. Operations-Schlüssel**).

L. Falls blklen mod t ≠ 0 ist, dann:
   a. Die Bytes mainblk+1 bis blklen werden per 'xor'-Operation **verknüpft** mit **und ersetzen** die Bytes mainblk+1–t bis blklen–t der letzten (**n**.) t-Byte-Gruppe:
      block[blklen–t–i] = block[blklen–t–i] xor block[blklen–i], i=0,...,(blklen mod t)–1.
   b. Der Datenblock wird durch Verschieben und Kopieren des Bytes vor dem letzten und weiterer Bytes davor zu einer **neuen t-Byte-Gruppe** erweitert:
      block[blklen+t–(blklen mod t)–i] = block[blklen–i], i=0,...,t–1.
   c. Die in Schritt 21.B gespeicherten Bytes des **1. Zufallszahlen-Prozesses** werden per 'xor'-Operation mit der neuen t-Byte-Gruppe verknüpft (analog zu Schritt 4.B).
   d. Die neue t-Byte-Gruppe wird via **inverse S-Tabelle** rückumgewandelt (s. Schritt 21.D).
   e. Die zuvor zur Erweiterung dienenden Bytes werden auf ihre Ausgangspositionen **zurückkopiert**:
      block[blklen–i] = block[blklen+t–(blklen mod t)–i], i= t–1,...,0.

M. Durch Erzeugung von Zufallszahlen des **1. Zufalls-Prozesse**s (Schritt 4.B) werden die Mantissenbytes dieser Zufallszahlen fortlaufend per 'xor'-Operation mit allen mainblk Datenbytes verknüpft.

N. Falls im Laufe der Bearbeitung der Zyklus des aktuellen 1. Pseudo-Zufallszahlen-Prozesses bereits einmal durchlaufen sein sollte, dann werden folgende Aktionen veranlasst:
   a. Der zuvor verwendete **Stellenwert** (**pval**) der t-Byte-Stelle im Operations-Schlüssel, ab der die letzten 2*7 Zeichen im Operations-Schlüssel zur Initialisierung der beiden Zahlen eines neuen Prozesses verwendet wurden, wird um (pval+7) modulo (t*256**(t–1) –7 +1) erhöht.
   b. Es wird ein neuer 1. Pseudo-Zufallszahlen-Prozess initialisiert (Schritt 4.A) und die so erzeugten Pseudo-Zufallszahlen werden weiter per 'xor'-Operation mit den restlichen der mainblk Datenbytes verknüpft (Schritt 4.B).

O. Die mainblk Datenbytes werden t-Byte-weise via **inverse S-Tabelle** rückumgewandelt.

Kurz-Untersuchung zur Sicherheit des Algorithmus

III. Einzelne Komponenten und Schritte

Die folgenden einzelnen/kombinierten Schrittnummern beziehen sich auf die Schritte im Dokument "Beschreibung des Algorithmus" oben, wobei zunächst der Standard-Fall betrachtet wird.

Das hier angewandte Chiffrier-Verfahren besteht im Kern aus 3 Komponenten, die kombiniert angewendet werden und voneinander abhängig sind:

1. Die erste Komponente stellt der Schlüssel bzw. der daraus abgeleitete Operations-Schlüssel dar. Diese Komponente dient auch als Ausgangspunkt für die Konstruktion der beiden anderen Komponenten.
2. Die zweite Komponente stellt die S-Tabelle bzw. beim Dechiffrieren der Daten die inverse S-Tabelle dar. Diese wird (bzw. beide Tabellen werden) aus dem Schlüssel bzw. aus dem aktuellen Operations-Schlüssel hergeleitet.
3. Die dritte Komponente bilden zwei fortlaufende Zufallszahlen-Prozesse, die ebenfalls durch den Schlüssel bzw. den Operations-Schlüssel initiiert werden.

4. Um den Wirkungsgrad der S-Tabellen speziell bei kurzen Schlüsseln zu erhöhen, wird die Schlüsselangabe auf die volle Länge von 256**t t-Byte-Gruppen erweitert, indem der Originalschlüssel mehrfach wiederholt und zuletzt auf 256**t t-Byte-Gruppen gekürzt wird.

5. Da die Eingabe eines Schlüssels von auch nur annähernd t*256**t Bytes einige Probleme verursachen kann, die Wiederholung des Originalschlüssels jedoch keine akzeptable Lösung darstellt, wurde mit dem in "Einleitung/Zusammenfassung", "Schlüssel(1)" und genauer in Schritt 2 beschriebenen Verfahren eine Methode integriert, die die Gleichförmigkeit bei der Generierung des Schlüssels durchbricht. Des weiteren eröffnet es die Möglichkeit, die Inhalte beliebiger Dateien als zusätzliche Quelle für die Schlüssel-Generierung zu verwenden.

6. Die Zufallszahlen in "Einleitung/Zusammenfassung", "Pseudo-Zufallszahlen-Prozesse" und in Schritt 4 sind für das „Rauschen“ zuständig, da Daten im allgemeinen nicht ungleichförmig sind. Erst die Ungleichförmigkeit ermöglicht, dass die S-Tabelle in weiten Bereichen und damit größtmöglich zur Wirkung kommt. Auch für eine entsprechende Wirkung der ‚xor’-Operation sind nicht zu gleichförmige Daten Voraussetzung.
   Darüber hinaus sind die Zufallszahlen jedoch auch für die (bzgl. der Varianten-Mächtigkeit allerdings nicht so bedeutende) „Grundsicherung“ der Daten zuständig, weshalb mit dem 2. Pseudo-Zufallszahlen-Prozess am Ende ("Einleitung/Zusammenfassung", "Definition der Chiffrierhülle" und Schritt 20.L) neben der S-Tabelle nochmals Pseudo-Zufallszahlen angewendet werden.
   Weiter steuern die Zufallszahlen-Bytes die Variabilität und die Zufälligkeit des Chiffrier-Prozesses. Das bedeutet varaible Blocklänge, Anzahl der Schleifendurchläufe, Verschiebewerte und Vergleichswerte werden, wie z.B. in den Schritten 4.A.d, 7, 17.B, 19, 20.I und 20.P durchgeführt, durch Zufallszahlen-Bytes erzeugt.
   Bei der Anwendung der Zufallszahlen gemäß Schritt 4.B wurde daher darauf Wert gelegt, dass die „Zufälligkeit“ nicht durch Randeffekte bei der Multiplikation in Mitleidenschaft gezogen wird. Daher werden nur die Kern-Bytes der jeweiligen (Pseudo-) Zufallszahlen verwendet, wobei Schritt 4.B.d nochmals eine Kopplung der beiden Zufallszahlen eines Prozesses sowie die Berücksichtigung auch der restlichen Kern-Bytes sicher stellt.

Des weiteren wird Wert darauf gelegt, dass ein Zufallszahlen-Prozess sich nicht wiederholt. D.h. immer dann, wenn dies der Fall sein sollte, wird zur Verhinderung von Angriffspunkten durch identische Operatoren und damit ggf. von ähnlichen oder identischen Blöcken auf einen neuen Zufallszahlen-Prozess umgeschaltet.
Tritt dieses Ereignis bezüglich des 1. Zufallszahlen-Prozess bei der Behandlung von Daten ein, so werden alle Komponenten neu generiert ("Einleitung/Zusammenfassung", "Komponenten-Wechsel(2)" und Schritt 18). Im Prinzip findet damit ein vollständiger Schlüsselwechsel vom bisherigen Schlüssel auf den aktuellen Operations-Schlüssel statt.

7. Die im Schritt 7 beschriebene Vorgehensweise wird sowohl für Operations-Schlüssel als auch, in abgewandelter Form (siehe Schritte 20.I und 21.J), für Daten verwendet. Eine Erklärung dazu wird in "Einleitung/Zusammenfassung", "Kern-Erklärung(2)" erläutert.
   Die Verschiebung in Schritt 7.F erschwert einen direkten statistischen Angriff bei der Operations-Schlüssel-Erschließung zusätzlich.
   Auch die Verknüpfung des Operations-Schlüssels mit den Daten (siehe Schritte 20.G und 20.J) garantiert den Einfluß jedes Bits des (aktuellen Operations-)Schlüssls auf jedes Bit eines Datenblocks.

8. S-Tabelle, Zufallszahlen sowie der eben beschriebene „Reißwolf“-Prozess werden nicht nur bei Datenblöcken sondern auch bei der Modifikation von Operations-Schlüsseln eingesetzt.
   Zur Erzeugung neuer S-Tabellen und inverser S-Tabellen (Schritte 12 und 18.B) sowie zur Erzeugung neuer Pseudo-Zufallszahlen-Prozesse (Schritte 13 und 18.B) werden jeweils neben dem neuen Operations-Schlüssel (Schritte 14 und 18.B) eigene Übergangs-Operations-Schlüssel erzeugt und verwendet (Schritte 11, 13 und 18.B), um ggf. Rückschlüsse und damit Angriffe auf Pseudo-Zufallszahlen, S-Tabellen und Operations-Schlüsseln bezüglich der Datenbehandlung durch diese Komponenten zu erschweren.
   Der Operations-Schlüssel (bzgl. Daten) wird während der Durchführung des Chiffrier-Verfahrens permanent auf die Daten angewendet (Schritte 20.G, 20.J, 21.I und 21.K).
   Um zu verhindern, dass es möglich erscheint, durch die wiederholte Anwendung des Operations-Schlüssels auf diesen vollständig rückzuschließen, wird dieser vor Erreichen der n-ten Anwendung modifiziert (Schritt 19). Dabei ist n die Anzahl der Byte-Gruppen im Operations-Schlüssel =256**t. Andererseits wird man bemüht sein, schon aus Aufwandsgründen nur möglichst notwendige Modifikationen durchzuführen und ggf. neue Angriffspunkte für die Operations-Schlüssel-Erschließung zu vermeiden.
   Durch den Bezug auf den 1. Pseudo-Zufallszahlen-Prozess (Schritt 19) wird eine Kopplung der Operations-Schlüssel-Entwicklung an die Entwicklung im 1. Pseudo-Zufallszahlen-Prozess hergestellt,und zwar nicht nur bzgl. des „wann“ sondern auch bzgl. des „wie“.

9. Allgemein wird die (inverse)S-Tabelle jeweils zwischen zwei benachbarten ‚xor’-Operationen angewendet, um die Mächtigkeit der möglichen S-Tabellen bei jeder geänderten Ausgangslage wieder neu ins Spiel zu bringen und die einzelnen Operations-Phasen gegeneinander „abzuschirmen“.

IV. Schlimmster-Fall-Szenarien

Schematisch werden bei der **Erzeugung der Komponenten** folgende Schritte ausgeführt:

A. 1. Übergangs-Operations-Schlüssel
B. S-Tabelle und inverse S-Tabelle
C. 2. Übergangs-Operations-Schlüssel
D. Pseudo-Zufallszahlen-Prozesse (Initialisierung; für Daten)
E. Ausgangs-Operations-Schlüssel

wobei **Operations-Schlüssel** wie folgt erzeugt werden (ohne Besonderheiten):

(1) 'xor' der Zufallszahlen des 1. Zufallszahlen-Prozesses (Schlüssel)
(2) S-Tabelle (B für C, D und E, B-Vorgänger für A)
(3) 'xor' der Schlüssel-Daten von links nach rechts
(4) S-Tabelle (B für C, D und E, B-Vorgänger für A)
(5) 'xor' der Schlüssel-Daten von rechts nach links
(6) S-Tabelle (B für C, D und E, B-Vorgänger für A)
(7) zyklische Verschiebung (gemäß Ergebnis von (4) und aktueller Zufallszahl)
(8) Wiederholung der Schritte (3) bis (7)
(9) 'xor' der Zufallszahlen des 2. Zufallszahlen-Prozesses (Schlüssel)
(10) S-Tabelle (B für C, D und E, B-Vorgänger für A)
(11) Wiederholung der Schritte (3) bis (8) mit (3) und (5) vertauscht.

Bei der **Chiffrierung pro Block** werden schematisch folgende Schritte ausgeführt, die bei Dechiffrierung in umgekehrter Reihenfolge mit inverser S-Tabelle durchlaufen werden:

a. S-Tabelle
b. ‚xor' der Zufallszahlen (1. Zufallszahlen-Prozess; Daten)
c. S-Tabelle
d. ‚xor' des Operations-Schlüssels (oder des 1./unteren Teils des Operations-Schlüssels)
e. S-Tabelle
f. ‚xor' der Daten von links nach rechts
g. S-Tabelle
h. ‚xor' der Daten von rechts nach links
i. S-Tabelle
j. ‚xor' des Operations-Schlüssels (oder des 2./oberen Teils des Operations-Schlüssels)
k. S-Tabelle
l. ‚xor' der Zufallszahlen (2. Zufallszahlen-Prozess; Daten)
m. S-Tabelle

Sei
- T die (Byte-Gruppen-)S-Tabelle,
- ZZ_1 die Zufallszahlen-Elemente des 1. Pseudo-Zufallszahlen-Prozesses in Blocklänge,
- ZZ_2 die Zufallszahlen-Elemente des 2. Pseudo-Zufallszahlen-Prozesses in Blocklänge,
- Op_A der/der 1. Teil des/der untere Teil des Operations-Schlüssel/s,
- Op_E der/der 2. Teil des/der obere Teil des Operations-Schlüssel/s,
- E_o obere (Byte-Gruppen-)Dreiecksmatrix auf allen nicht Null Positionen mit Wert 1,
- E_u untere (Byte-Gruppen-)Dreiecksmatrix auf allen nicht Null Positionen mit Wert1,
- $\oplus$ die Byte-Gruppen -weise Addition ohne Übertrag (‚xor'),
- $\otimes$ die zu $\oplus$ gehörende (Vektor-)Multiplikation,

dann lässt sich die Block-Chiffrierung auch wie folgt beschreiben:

$$cBlock := T(ZZ_2 \oplus T(Op_E \oplus T(T(T(Op_A \oplus T(ZZ_1 \oplus T(Block))) \otimes E_o) \otimes E_u)))$$

Die im folgenden beschriebenen Szenarien sind zunächst fiktiv, d.h. ob sie so in der Realität auftreten können oder nicht, ist erst einmal nicht von Bedeutung. Vielmehr ist es Absicht, die hohe Sicherheit des Systems zu demonstrieren, selbst wenn – wie auch immer – bestimmte Komponenten als „bekannt“ unterstellt werden.

Natürlich wird bei allen Szenarien der Ausgangs-Schlüssel als nicht bekannt angenommen.

10. Betrachten wir das Szenario, bei dem sowohl die Zufallszahlen-Prozesse für die Daten als auch die S-Tabelle als auch die Ausgangs-Daten sowie die chiffrierten Daten bekannt sind. Der **Ausgangs-Operations-Schlüssel** sei hierbei **unbekannt**.
    1. Dann kommt man von des Ausgangsdaten bis zum Schritt d, von den chiffrierten Daten über die Schritte m, l und k zum Schritt j.
    2. Auf Grund der S-Tabelle kann zwar ein großer Teil des 1. Übergangs-Operations-Schlüssels hergeleitet werden, aber ein nicht unerheblicher Teil bleibt offen.
    3. Schon der 2. Übergangs-Operations-Schlüssel ist nicht mehr direkt ableitbar, da in Schritt (5) die nicht bekannten Informationen des 1. Übergangs-Operations-Schlüssels massiv den bekannten Anteil beeinflussen. Auch Schritt (7) für den 2. Übergangs-Operations-Schlüssel ist unbekannt, da hier die höchste Byte-Gruppe des 1. Übergangs-Operations-Schlüssels eingeht, die ja gerade nicht hergeleitet werden konnte. Hierfür könnten jedoch ggf. noch alle möglichen Varianten durchgespielt werden. Spätestens aber mit der Wiederholung der Schritte (3) bis (7) (Schritt (8) und nochmals 2-mal in Schritt (11)) ist der dann 2. Übergangs-Operations-Schlüssel nicht mehr nachvollziehbar.
    4. Die Komponente D ist zwar bekannt, aber hierüber lassen sich nur wenige Elemente des 2. Übergangs-Operations-Schlüssels herleiten. Spätestens mit wiederholter Durchführung der Schritte (3) und (5) auf dem 2. Übergangs-Operations-Schlüssel ist der Ausgangs-Operations-Schlüssel nicht mehr direkt oder indirekt herleitbar.
    5. Damit kann nicht nachvollzogen werden, wie man ohne Überprüfung sämtlicher Operations-Schlüssel-Zustände von Schritt d zu Schritt j bzw. umgekehrt kommen sollte. Auch analytische/statistische Untersuchungs-Versuche über mehrere bzw. alle Datenblöcke sind zum Scheitern verurteilt, da noch bevor die Möglichkeit nur hypothetisch gegeben wäre, den Operations-Schlüssel zu erschließen, dieser durch die Schritte (3) bis (7) oder in der Reihenfolge (5), (6), (3), (4), (7) modifiziert wird. Dabei kann selbst die Kenntnis des „wann“ und „wie“ durch Kenntnis der Zufallszahlen-Prozesse nicht weiter helfen, da die Schritte (3) und (5) kombiniert mit (4) bzw. (6) jedwede Form von Teilerkenntnissen wieder zu Nichte machen.
    6. Ergebnis: Der Ausgangs-Operations-Schlüssel ist analytisch bzw. mit statistischen Mitteln (soweit z.Zt. ersichtlich) nicht erschließbar.

11. Betrachten wir das Szenario, bei dem sowohl die Zufallszahlen-Prozesse als auch der Ausgangs-Operations-Schlüssel als auch die Ausgangs- sowie die chiffrierten Daten bekannt sind. Die **S-Tabelle** seien hierbei **unbekannt**.
    1. Ohne S-Tabelle sind Ausgangs- und chiffrierte Daten zunächst nicht direkt modifizierbar.
    2. Betrachtet man obige Schritte vom Ausgangs-Operations-Schlüssel zum 2. Übergangs-Operations-Schlüssel rückwärts, so wird sofort ersichtlich, dass spätestens bei Schritt (6) die Rückverfolgbarkeit zum 2. Übergangs-Operations-Schlüssel endet.
    3. Thesen zur Belegung einzelner Elemente in der S-Tabelle helfen nicht weiter, da mit Schritt (5) und (3) jeweils eine wechselseitige Verknüpfung mit den restlichen S-Tabellen-Ergebnissen stattfindet, die dann wieder Basis für die nächste bzw. vorausgehende Operation ist.

4. Selbst wenn nur ein Bruchteil der S-Tabelle tatsächlich Verwendung findet, helfen diese Annahmen nicht weiter, da unbekannt bleibt, welche Teile dies sind.
5. Der bekannte Pseudo-Zufallszahlen-Prozess ermöglicht nur die Erschließung weniger Stellen des 2. Übergangs-Operations-Schlüssels. Damit dann aber auf den 1. Übergangs-Operations-Schlüssel rückzuschließen, trifft jedoch auf die gleichen Hindernisse, wie beim Versuch vom Ausgangs- auf den 2. Übergangs-Operations-Schlüssel zu schließen.
6. Ergebnis: Die S-Tabelle ist analytisch bzw. mit statistischen Mitteln (soweit z.Zt. ersichtlich) nicht erschließbar.

12. Betrachten wir das Szenario, bei dem sowohl die S-Tabelle als auch der Ausgangs-Operations-Schlüssel als auch die Ausgangs- sowie die chiffrierten Daten bekannt sind. Die beiden **Pseudo-Zufallszahlen-Prozesse** für Daten seien hierbei **unbekannt**.
    1. Betrachtet man die Chiffrierung der Datenblöcke, so stellt man fest, dass trotz Kenntnis von S-Tabelle und inverser S-Tabelle sowie dem Operations-Schlüssel zunächst bei den Schritten b und l Ende des Versuchs ist, etwas über die Zufallszahlen-Prozesse zu erfahren.

       Auf Grund der Schritte f und h lassen sich einzelne Stellen innerhalb eines Datenblocks nicht separieren (die ‚xor'-Operation wirkt in diesem Umfeld wie eine Art Restklassen-Bildung), d.h. die beiden Zufallszahlen-Prozesse müssten dem gemäß vollständig enumerativ ausgeführt werden, um sie zu ermitteln.
    2. Wollte man die Pseudo-Zufallszahlen-Prozesse (Komponente D) aus den Operations-Schlüsseln erschließen, so müsste der 2. Übergangs-Operations-Schlüssel zumindest bezüglich der für die Prozesse relevanten Stellen bekannt sein.
    3. Auf Grund der S-Tabelle kann zwar ein großer Teil des 1. Übergangs-Operations-Schlüssels hergeleitet werden, aber ein nicht unerheblicher Teil bleibt offen.
    4. Der 2. Übergangs-Operations-Schlüssel ist daraus nicht mehr direkt ableitbar, da in Schritt (5) die nicht bekannten Informationen des 1. Übergangs-Operations-Schlüssels massiv den bekannten Anteil beeinflussen. Auch Schritt (7) für den 2. Übergangs-Operations-Schlüssel ist unbekannt, da hier die höchste Byte-Gruppe des 1. Übergangs-Operations-Schlüssels eingeht, die ja gerade nicht hergeleitet werden konnte. Hierfür könnten jedoch ggf. noch alle möglichen Varianten durchgespielt werden. Spätestens aber mit der Wiederholung der Schritte (3) bis (7) (Schritt (8) und nochmals 2-mal in Schritt (11)) ist der dann 2. Übergangs-Operations-Schlüssel auch in Teilen nicht mehr nachvollziehbar.
    5. Der Versuch, die Zufallszahlen-Prozesse aus dem Ausgangs-Operations-Schlüssel herzuleiten, trifft in Schritt (9) sofort auf das Problem, den Zufallszahlen-Prozess für den Übergang vom 2. Übergangs- zum Ausgangs-Operations-Schlüssel zu ermitteln. Zwar ist, wie in Schritt 13 beschrieben, die Stelle im 2. Übergangs-Operations-Schlüssel bekannt, ab der die 14 Zeichen zur Erzeugung des hier betrachteten Zufallszahlen-Prozesses Verwendung finden, die Zeichen selbst aber gehören zum 2. Übergangs-Operations-Schlüssel, der ja unbekannt ist. Ohne diesen Zufallszahlen-Prozess sind aber weder die Schritte (1) und (9), noch die Schritte (7) und damit (8) nachvollziehbar, von dem Problem, die Schritte (3) und (5) sowie deren Wiederholungen in Schritt (8) „umzukehren", ganz zu schweigen.
    6. Ergebnis: Die Pseudo-Zufallszahlen-Prozesse für Daten sind analytisch bzw. mit statistischen Mitteln (soweit z.Zt. ersichtlich) nicht erschließbar.

13. Weitere Szenarien, z.B. die Kenntnis des 1. Übergangs-Operations-Schlüssels usw., sind denkbar jedoch nicht unbedingt sinnvoll. Bei den aufgezeigten Szenarien ging es darum, anhand bekannter bzw. ggf. bekannter Elemente wie chiffrierte Daten und Ausgangs-Daten auf der einen Seite und ggf. durch Datenanalyse auch mit statistischen Methoden vielleicht zu erlangender Erkenntnisse zu S-Tabelle, Pseudo-Zufallszahlen-Prozessen und Ausgangs-Operations-Schlüssel auf der anderen Seite nachzuweisen, dass dennoch eine hohe Sicherheit gewährleistet werden kann.

14. Ergebnis obiger Kurz-Untersuchungen des Standard-Falls ist, dass man zum Nachvollziehen des gesamten Chiffrierprozesses entweder den Ausgangs-Schlüssel oder tatsächlich sämtliche der drei genannten Komponenten (B, D, E) benötigt. **Zwei der genannten Komponenten reichen nicht aus, um auf die fehlende Komponente zu schließen.**

15. Selbst wenn sämtliche der Komponenten B, D und E und damit der gesamte Chiffrierprozess bekannt sein sollte, so ist anhand der nicht bekannten S-Tabelle, der unbekannten Pseudo-Zufallszahlen-Prozesse sowie dem „Reißwolf“-Prozess, die sämtlich zunächst aus dem Ausgangs-Schlüssel hervorgehen, klar, dass sich der Ausgangs-Schlüssel wird kaum direkt aus genannten Komponenten ermitteln lassen.
    Der Ausgangs-Schlüssel ist also selbst gegenüber seinem Chiffrierprozess wirkungsvoll „abgeschirmt“.

16. Als Konsequenz der Untersuchung kann ebenfalls angesehen werden, dass der Schlüsselraum nicht nur als auf die S-Tabelle reduziert angesehen werden kann. Wirksam sind tatsächlich alle 256**t Stellen des Schlüssels, d.h. t*256**t Bytes. Der Schlüsselraum umfasst daher (256**t)**(256**t) Elemente, die mindestens zu einem größeren Teil durch betrachtete Algorithmen angesprochen werden können.
    Ob der vollständige Parameterraum gemäß "Einleitung/Zusammenfassung", " Parameterraum-Größe" angesprochen werden kann, indem man beim Übergang zum nächsten Prametersatz nur einzelne aber nicht alle Komponenten ändert, bleibt weiteren Untersuchungen vorbehalten.

V. Notwendigkeit der angegebenen Komponenten und Schritte

Dass die angegebenen Komponenten und Schritte nicht „überflüssig“ sind bzw. nicht beliebig reduziert werden können, kann durch folgende Überlegungen begründet werden:

(a) Die Anwendung der Block-Chiffrierschritte f. bis i. alleine reicht nicht aus, um bei gleichen Ausgangsblöcken gleiche Zielblöcke zu vermeiden. Vielmehr ist es notwendig, einen fortlaufenden „Zufalls“-Prozess in die Zielblock-Erzeugung einzubeziehen, z.B. durch Anwendung eines permanent fortgeschalteten Operations-Schlüssels oder eines Pseudo-Zufallszahlen-Prozesses.

(b) Wird nur ein Operations-Schlüssel verwendet, muss dieser zu Anfang der Block-chiffrierung auf **alle** Blockelemente angewandt werden und zwar so, dass nicht mehrere Blockelemente mit demselben Operations-Schlüssel-Element oder einzelne Blockelemente gar nicht behandelt werden. Wird dies nicht beachtet, so können die Schritte f. bis i. bei identischen Daten-Elementen entweder 4-fach alternierende oder alternierende oder konstant identische Werte liefern.

(c) Die Anwendung des Operations-Schlüssels lediglich am Ende oder teilweise am Ende ist **nicht** ausreichend, da auch hier unabhängig von der S-Tabelle T nach den Schritten f. bis i. jedes 4. Element gleich **T**(0) wäre und damit ein Teil der Operations-Schlüssel-Elemente y als „**T**(**T**(0) **xor** y)“ zu 1/4 bekannt bzw. zu ermitteln wären.

(d) Einen nicht vollständig überdeckenden Operations-Schlüssel am Anfang mit einem Pseudo-Zufallszahlen-Prozess am Ende zu verwenden ist ebenfalls nicht ausreichend, da hier ebenfalls analog zum eben gesagten 1/4 der Pseudo-Zufallszahlen-Bytes und damit ggf. der gesamte Pseudo-Zufallszahlen-Prozess erschlossen werden kann.

(e) Allgemein muss daher bei alleiniger Verwendung des Operations-Schlüssels als „Zufalls“-Prozess die Länge des Operations-Schlüssels mindestens die Länge eines Blocks umfassen. Des weiteren muss der Operations-Schlüssel jeweils nach seiner Abarbeitung erneut modifiziert werden, wobei die Modifikation nicht so geschehen darf, dass z.B. bei binären Nullen als Daten identische Zielblöcke oder sogar Zielblöcke entstehen, die ganz oder teilweise mit einzelnen Operations-Schlüsseln identisch sind.

(f) Wird nur ein Pseudo-Zufallszahlen-Prozess verwendet, so ist dieser analog zum eben gesagten nur am Anfang einzusetzen. Auch hier muss dieser Prozess **alle** Blockelemente modifizieren.

(g) Falls man einen „Zufalls“-Prozess nur am Anfang einsetzt, so bleibt die Gefahr, dass dieser „ermittelt“ werden kann, wenn die Zieldaten zufällig (wenn auch nur mit extrem geringer Wahrscheinlichkeit) sehr homogen sein sollten. Zur Bestätigung dieser Aussage sind die zuvor vorgetragenen Argumentationsketten unter Vertauschung der Reihenfolge und mit inverser S-Tabelle anwendbar. Um dies zu vermeiden, sind sowohl ein „Zufalls“-Prozess am Anfang als auch ein "Zufalls"-Prozess am Ende notwendig, die beide jeweils den Block voll überdecken müssen.

(h) Da der Operations-Schlüssel als Ausgangspunkt für die Erneuerung eines Chiffrierprozesses dient, falls z.B. der (Haupt-)Pseudo-Zufallszahlen-Prozess periodisch zu werden droht, ist seine Erzeugung auf jeden Fall notwendig, auch wenn dieser nicht zur Erzeugung eines „Zufalls"-Prozesses eingesetzt wird. Ebenfalls ist der Kompromittierungsschutz vergangener Daten des Systems selbst bei Kompromittierung aller aktuellen System-Komponenten mit dieser Erneuerung des Chiffrierprozesses und damit mit dem Operations-Schlüssel verbunden (siehe IV., a) unten). Des weiteren repräsentiert der Operations-Schlüssel in gewisser Weise auch die Mächtigkeit der verwendeten S-Tabelle und ist daher ein wesentlich besserer Schutz als ein 14 Bytes umfassender Pseudo-Zufallszahlen-Prozess.

(i) Als offener Punkt bleibt jedoch die Periodizität eines solcherart modifizierten Operations-Schlüssels. Nach Untersuchungen des Autors gibt es unter den S-Tabellen mit n=2**m Elementen nur n unter den n! möglichen Tabellen, die bzgl. der 'xor'-Operation voll assoziativ sind, d.h. **T**(y **xor** z) = **T**(y) **xor** z für alle y,z.

Hier sind sicher noch weitere Untersuchungen nötig (falls nicht bereits geschehen), um ein klareres Bild der Periodizität von so erzeugten Operations-Schlüsseln zu gewinnen.

Da aber die Periodizität eines unter Anwendung einer S-Tabelle permanent fortgeschalteten Operations-Schlüssels zunächst als ungeklärt verbleibt, wurden im vorliegenden Algorithmus beide „Zufalls"-Prozesse – bzgl. des Operations-Schlüssels in abgewandelter Form (siehe Schritt 19) – in den Schritten a. bis e. und j. bis m. integriert, womit eine „sehr hohe" Zykluslänge gekoppelt mit einem hohen Kompromittierungsschutz sicher gestellt werden kann.

VI. Erweiterungen zur Erhöhung der Sicherheit

Zur Erhöhung der Sicherheit speziell für ein professionelleres Umfeld bieten sich weitere Modifikationen des bisher vorgestellten Standard-Algorithmus an, deren Vorteile im weiteren genauer erörtert werden:

1) Der Wechsel der Komponenten B, D und E findet beim Standard-Algorithmus nur dann statt, wenn der 1. Pseudo-Zufallszahlen-Prozess einen Zyklus durchlaufen hat. Statt einfacher Modifikation des aktuellen Operations-Schlüssels ist ein vollständiger Wechsel dieser Komponenten vorgesehen (Erweitert 1). Eine spezielle Variable (hier: 'SignBits') legt dabei fest, nach wie vielen Datenblöcken im Durchschnitt ein kompletter Komponenten-Wechsel durchzuführen ist (Schritt 18/19).

2) Für jeden Block ist ein eigener Operations-Schlüssel-Abschnitt bzw. ein eigener Operations-Schlüssel zu verwenden (Erweitert 1). Der Operations-Schlüssel wird nach Abarbeitung jeweils gemäß den Schritten (3) bis (8) oder dem Schritt (11) modifiziert (Schritt 19).

3) Die Operations-Schlüssel-Schritte (8) und (11) werden n-fach wiederholt (Erweitert 1, Schritt 7), wobei sich n aus dem jeweiligen Pseudozufallszahlen-Prozess sowie dem aktuell verwendeten Operations-Schlüssel ergibt mit einer Begrenzung nach oben und unten

4) Die Blockchiffrier-Schritte f bis i werden um einen analog Schritt (7) ergänzten Schritt (Erweitert 1, Schritt 20.I.e) m-fach wiederholt (Erweitert 1, Schritt 20.I), wobei sich m aus dem jeweiligen Pseudozufallszahlen-Prozess sowie dem verwendeten aktuellen Operations-Schlüssel ergibt mit einer Begrenzung nach oben (2***mf**) und unten (Festwert **mf**).

5) Es wird ein eigener Generierungs-(Erweitert 1.1)/Modifikations-(Erweitert 1.1.1) Schlüssel zur Änderung des Operations-Schlüssels angewendet. Dabei werden die aktuellen Komponenten gemäß B, D und E dieses separaten Schlüssels zur Erzeugung eines jeweils neuen Operations-Schlüssels durch Chiffrierung des aktuellen Operations-Schlüssels bei Komponenten-Wechsel gemäß 1) bzw. bei Operations-Schlüssel-Modifikation gemäß 2) eingesetzt.

6) Es findet eine zyklische Byte-Verschiebung jedes Datenblocks nach Chiffrierung statt (Erweitert E, Schritt 20.P).

Die genannten Erweiterungen sind durch folgende Überlegungen motiviert:

a) Da jeder Komponenten-Wechsel wie eine Ausgangs-Initialisierung mit dem jeweiligen Operations-Schlüssel statt dem Ausgangs-Schlüssel durchgeführt wird, sind ausgehend von den aktuellen Komponenten keinerlei „einfache“ Rückschlüsse auf vorangehende Komponenten möglich.
Sollte ein Komponentensatz kompromittiert werden, so sind die Datenblöcke **vorangehender** Komponentensätze **nicht** kompromittiert.
Durch Modifikation 1) wird mit Hilfe der speziellen Variablen (’SignBits’) die diesbezügliche „Sicherheit“ des Chiffrierverfahrens festgelegt. Im Extremfall kann nach jedem Datenblock ein Komponenten-Wechsel stattfinden.
Zur Reduktion des Aufwands können zwischen den Komponenten-Wechseln „einfache“ Operations-Schlüssel-Modifikationen gemäß 2) durchgeführt werden, was jedoch bei Kompromittierung eines Komponentensatzes keinerlei abgrenzende Sicherheit für nachfolgende Blöcke bietet.

b) Um die Wirksamkeit der S-Tabellen auf größere Tabellenbereiche auszudehnen (speziell bei „kleinen“ Blocklängen im Verhältnis zur Tabellengröße) und die Durchmischung der Daten sowohl bei der eigentlichen Blockchiffrierung als auch bei der Komponenten-Erzeugung bzw. bei der Operations-Schlüssel-Modifikation zu erhöhen, werden die entsprechenden Linearoperationen gemäß Modifikation 3) und 4) m-fach durchlaufen.
Falls die beteiligten Zyklenlängen der jeweiligen S-Tabelle nicht größten Teils kleiner sind als der Wiederholungsfaktor m, wird ein Block, der zunächst nur aus genau einem (entsprechend oft wiederholten) Element besteht, nach $n=\log_2$(Anzahl von t-Byte-Gruppen Elementen im Block)/2 Runden wieder auf einen Block abgebildet, der die für seine Größe durchschnittliche Anzahl unterschiedlicher Elemente besitzt (2-malige Verdoppelung der unterschiedlichen Elemente pro Runde bei „geeigneter“ S-Tabelle).

c) Um bei Kompromittierung eines Komponentensatzes zu verhindern, dass der nächste Komponentensatz und die damit chiffrierten Datenblöcke ebenfalls kompromittiert werden, kann **bei Stromchiffrierung** gemäß Modifikation 5) ein Operations-Schlüssel - Generierungs- bzw. -Modifikations-Schlüssel eingeführt werden, mit dessen Hilfe eine fortlaufende Kette von Generierungs-Komponenten gemäß A bis E erzeugt wird, die via Komponentenwechsel-Parameterdatei bereit gestellt nur dazu dienen, den Operations-Schlüssel am Anfang eines Komponenten-Wechsels bzw. eines Operations-Schlüssel-Wechsels als Pseudo-Datenblock zu chiffrieren, um die eigentlichen Schritte A bis E des Komponenten-Wechsels bzw. die Schritte (3) bis (8) oder den Schritt (11) mit diesem „chiffrierten“ Operations-Schlüssel als Ausgangspunkt vorzunehmen.
Sollte ein Komponentensatz unter Einsatz dieses Verfahrens kompromittiert werden, so sind auch die Datenblöcke **nachfolgender** Komponentensätze **nicht** kompromittiert.
Um diese „Serie“ von Generierungsschlüssel-Komponenten zu kompromittieren, müssten zunächst „sehr viele“ normale Komponentensätze kompromittiert sein oder der Generierungsschlüssel-Komponentensatz ist ab einem bestimmten Zeitpunkt „bekannt“ (der Generierungsschlüssel selbst ist nicht bekannt, wenn nur mit den Komponenten selbst gearbeitet wird; im übrigen müssen die Ausgangs-Komponenten natürlich nicht von **einem** Schlüssel stammen).
Damit wären zwar die Datenblöcke ab diesem Zeitpunkt bekannt, Datenblöcke, die mit vergangenen Komponenten behandelt wurden, blieben aber auch in diesem Fall **nicht kompromittiert.**

d) Organisiert man den Nachrichtenverkehr so, dass zum einen eine „Zentrale“ und andererseits „Satelliten“ existieren, so kann jedem Satelliten ein eigener Generierungsschlüssel zugeordnet werden, d.h. die verschiedenen Nachrichtenwege sind individuell und unabhängig gesichert.

Da der Chiffrierverfahrens-Fortschritt **nicht** an Blockinhalte gekoppelt ist und **nicht** davon abhängt, ob eine Chiffrierung oder eine Dechiffrierung mit dem aktuellen Parametersatz stattfindet, kann eine (gespreizte) Stromchiffrierung auch in beide Richtungen gleichzeitig (jedoch für **einen** Parametersatz nicht parallel) durchgeführt werden, soweit man sich jeweils auf eine Reihenfolge einigt. Die Komponenten- und Parameter-Entwicklung läuft in diesen Fällen im jeweiligen Satelliten und in der Zentrale absolut parallel mit der genannten hohen Sicherheit und **ohne einen weiteren Schlüsselaustausch.**

Da Parameterdateien intern so ausgelegt sind, dass
- alle Zustände einer Parameterdatei gültige reguläre Zustände sind,
- verschlüsselte und unverschlüsselten Parameterdateien nicht unterschieden werden können und damit
- jede auch „falsch“ entschlüsselte eine funktionsfähige Parameterdatei ist,

kann ein Satellit eine Parameterdatei statt mit dem regulären Schlüssel mit einem **„Kompromittierungs“-Schlüssel** „entschlüsseln“, so dass die Verwendung einer solcherart behandelten Parameterdatei der Zentrale die „Kompromittierung“ des Satelliten anzeigt, **ohne dass ein Dritter irgend eine technische Möglichkeit hätte, dies festzustellen.** Damit kann die Zentrale ab diesem Zeitpunkt selbst auf die so erzeugte „Kompromittierungs“-Parameterdatei übergehen und dem Satelliten „unwichtige“ Informationen übermitteln, um ggf. mitlesende Dritte zu täuschen bzw. auf falsche Fährten zu führen, ohne dass dies direkt entdeckt werden könnte.

e) Der gesamte hier besprochene **Nachrichtenaustausch** kann **via Internet** stattfinden. Da ein fehlerhafter Datenblock andere Datenblöcke und deren Chiffrierung bzw. Dechiffrierung **nicht** beeinflusst, ist eine Wiederholung von Inhalten und/oder von Blöcken jeder Zeit durchführbar, wobei der Inhaltswiederholung Vorrang gegeben werden sollte, falls die beschriebene Kompromittierungssicherheit in vollem Umfang beibehalten werden soll.

f) Um statistische Angriffe bei fester Blocklänge zu erschweren bzw. unmöglich zu machen, kann es sinnvoll sein, eine zyklische Byte-Verschiebung gemäß Modifikation 6) vorzunehmen.

Erweiterung und Anwendung des Algorithmus

Beispiel der Anwendung des Algorithmus im Agenten-Umfeld

Vorausgesetzt sei, dass die gesamte Kommunikation mit CODING als Strom-Chiffrierung unter Verwendung von „normalen“ und Komponentenwechsel- Parameterdateien durchgeführt wird und jeder Agent seinen eigenen Satz von Parameterdateien besitzt, wie auch immer er dazu gekommen ist. Des weiteren wird vorausgesetzt, dass jede Kommunikation zwischen Agenten über eine (oder mehrere) Zentrale(n) läuft. Eine direkte Kommunikation zwischen Agenten ist nicht sinnvoll und aus der Sichtweise von Kompromittierungen einzelner Agenten gefährlich, von technischen Problemen in diesem Zusammenhang ganz zu schweigen.

Zunächst muss am Ende einer offline-Bearbeitung zusätzlich ein Komponentenwechsel vorgenommen werden.

Falls dann ein Agent zwischen zwei offline-Bearbeitungen entdeckt wird, so hat das bzgl. der Sicherheit zunächst keinerlei Konsequenzen, da die beim Agenten zu findenden Parameterdateien den Zustand direkt **nach** einem Komponentenwechsel bei Strom-Chiffrierung repräsentieren, so dass der Agent, falls er zur Aktivierung der Parameterdateien gezwungen werden sollte, einfach seinen Kompromittierungsschlüssel verwendet und die Zentrale daher informiert ist. Die bisherigen Blöcke können selbst mit den korrekten ("normalen") Schlüsseln vom Agenten nicht mehr dekodiert oder reproduziert werden (!), wie in "Kurz-Untersuchung zur Sicherheit des Algorithmus" gezeigt wurde.

Um also direkt an sicherheitsrelevante Informationen zu gelangen, müsste der Gegner die Parameterdateien kopieren und sich dann beim Versandt des folgenden Blocks (d.h. des ersten Blocks bezüglich der kopierten Parameterdateien) diesen ebenfalls kopieren, um mit intelligenten „brut force“-Angriffen die Schlüssel der Parameterdateien zu knacken, wobei der erste Block nur zur Überprüfung dient, ob die richtigen Schlüssel gefunden wurden.

Bezüglich des bisher realisierten Algorithmus stört daher die Schwachstelle des „einfachen“ Schlüssels zur Sicherung der Parameterdateien. Da ein solcher Schlüssel ja bewusst einfach sein soll, sollte auch hier zusätzlich auf den Einsatz von Überlagerungsdateien zurückgegriffen werden. Damit wären Parameterdateien gegenüber intelligenten „brut force“-Angriffen wesentlich besser geschützt, falls der Gegner diese an sich bringen kann.

Selbst wenn der Gegner die Möglichkeit haben sollte, an die Parameterdateien heranzukommen, verbleibt das Problem, dass der Agent davon nichts mitbekommen darf. Denn würde der Agent etwas merken, so würde er sicher keinen regulären Block sondern nur Kompromittierungsblöcke versenden, womit für den Gegner keinerlei relevante Informationen mehr zu gewinnen wären. Des weiteren gibt es für den Gegner das Problem, dass er sämtliche Parameterdateien zum gleichen Zeitpunkt kopieren muss. Wird eine Parameterdatei nicht kopiert, bevor sie wieder regulär verwendet wird, sind alle bereits kopierten Parameterdateien wertlos (!).

Für die Handhabung heißt das, dass sich die verwendeten Parameterdateien möglichst auf unterschiedlichen Medien befinden sollten und nur zum Chiffrieren zusammenzubringen sind. Z.B. wären die Festplatte des Chiffrierungsrechners, ein Memory-Stick, den der Agent immer bei sich trägt, und ein drittes wiederbeschreibbares Medium, das entweder in einem Safe liegt oder z.B. in der Wohnung des Agenten versteckt wird, geeignet. Gehen wir weiter davon aus, dass der Agent

sich normalerweise jeden Tag bei der Zentrale meldet, dann verbleiben dem Gegner ca. 24 Stunden, um alle auf diesen drei Medien gespeicherten Informationen so zu kopieren, dass der Agent davon nichts mitbekommt (!). Kommt er auch nur ansatzweise auf die Idee, dass etwas nicht stimmt, wird er keinen regulär chiffrierten Block mehr erzeugen, ohne dass der Gegner das ermitteln kann, womit die Parameterdateien wertlos sind (!).

Für den vorliegenden Algorithmus heißt das, dass ggf. eine dritte Parameterdatei integriert werden sollte, um das hier genannte dritte Medium zu ermöglichen.

Umgekehrt ist es aber auch zwingend notwendig, dass der Agent beim geringsten Zweifel seine Kompromittierung anzeigt. Kann der Gegner nämlich sämtliche Parameterdateien und den zugehörigen ersten Block an sich bringen, kann er, statt „brut force“-Angriffen, auch versuchen, den Agenten zur Herausgabe der korrekten Schlüssel zu nötigen, da der Gegner die Schlüssel-angaben des Agenten anhand dieses ersten Blocks jeder Zeit überprüfen kann (!).

Hier zeigt sich, dass der offline-Betrieb sich durchaus als notwendig herausstellen könnte. Im online-Betrieb könnte der Gegner nach Versandt des ersten Blocks z.B. „einfach“ den Strom unterbrechen und hätte alle benötigten Parameterdateien ggf. noch unverändert vorliegen, soweit nicht nach jedem Block ein Komponentenwechsel durchgeführt und diese Komponenten sofort in die Parameterdateien gespeichert werden. Natürlich lässt sich das durch Batterie-Betrieb vermeiden, aber der Gegner könnte den Agenten genau nach Versandt des ersten Blocks einfach überfallen.

Im offline-Betrieb ist eine solche Vorgehensweise schon schwieriger zu realisieren. Hier gibt es nach außen zunächst keine Anhaltspunkte (falls nicht gerade der Bildschirminhalt des Chiffrier-rechners mitgelesen werden kann; sollte das der Fall sein, dann könnte der Feind die Informationen gleich direkt lesen, ohne Schlüssel !) und die zu versendenden Informationen können zum einen Zeitpunkt erzeugt und zu einem anderen Zeitpunkt versandt werden. Zum Sendezeitpunkt wäre es für den Gegner zunächst zu spät, denn die Parameterdateien sind zu diesem Zeitpunkt bereits wieder verschlüsselt. Mit weiteren hier nicht erörterten Verfeinerungen lässt sich die diskutierte Problemstellung auch für den online-Fall (!) ohne Eingriffsmöglichkeiten des Gegners sicher auslegen (es sein denn, der Gegner hat die Möglichkeit den Rechner unmittelbar zu stoppen und den Hauptspeicher auszulesen).

Für die Dechiffrierung der Informationen der Zentrale müsste der Agent im offline-Betrieb jedoch erneut die Parameterdateien aktivieren und damit die regulären Schlüssel verwenden. Damit könnte der Gegner wieder, jetzt zu diesem Zeitpunkt, zuschlagen. Doch hier ist die Situation für den Agenten „unproblematisch“, falls ein unabhängiger zweiter Parameterdatei-Satz nur für den Empfang der Informationen der Zentrale Verwendung findet. Die regulären Schlüssel für diese Parameterdateien kann der Agent ruhig herausgeben, soweit er sich sicher sein kann, dass er die nächste Anfrage mit seinem Kompromittierungsschlüssel sendet, worauf die Zentrale seinen Kompromittierungs-Zustand erkennen kann und keine „geheimen“ Informationen mehr sendet. Vergangene Empfangs-Blöcke sich auf keinen Fall damit kompromittiert (!; siehe die Untersuchung zur Sicherheit).

Nicht vermieden werden kann, dass die letzte Information der Zentrale, falls sie einmal vom Agenten empfangen wurde, dem Gegner bekannt werden kann, da die Entschlüsselung dieser Information unter voller Kontrolle des Gegners erzwungen werden kann. Zwar sind entsprechende Manipulationen durch den Agenten denkbar, aber unter der angenommenen hohen Aufmerksamkeit und den technischen Möglichkeiten des Gegners sind diese wohl nicht durchführbar, zumindest aber sehr zweifelhaft.

Stellt der Agent fest, dass er ankommende Informationen der Zentrale nicht mehr dechiffrieren kann, so sendet er der Zentrale mit der nächsten Anfrage seinen Stand der Parameterdateien ankommender Informationen. Da diese verschlüsselten Dateien überchiffriert werden und „nur“ den beim Agenten ankommenden Informationsverkehr betreffen, ist dies unproblematisch. Die Zentrale wird diese ankommenden Parameterdateien analysieren und ihre Parameterdateien auf den Stand der Dateien des Agenten zurücksetzen bzw. korrigieren. Bleibt die Störung bestehen, kann der Agent die Zentrale über diesen Sachverhalt informieren und die gestörte ankommende Kommunikation zur Kenntnis nehmen. D.h. er kann keine weiteren Informationen mehr von der Zentrale empfangen aber immer noch senden und er kann gleich oder später nochmals versuchen diese Situation zu ändern.

Dieses Verhalten kann der Gegner nicht weiter als bisher beschrieben verwerten, denn die zuvor beschriebenen Parameterdateien betreffen „nur“ die beim Agenten ankommenden Informationen.

Kann umgekehrt die Zentrale trotz entsprechender Bemühungen keine Resynchronisation mit den sendeseitigen Parameterdateien des Agenten durchführen, so wird sie dem Agenten ihre empfangsseitigen Parameterdateien mit der nächsten Rückmeldung zur Verfügung stellen, die der Agent seinerseits sendeseitig einsetzen kann. Auch dies ist „unproblematisch“, da es sich um chiffrierte Parameterdateien handelt, die unter „normalen“ Umständen auch beim Agenten so vorgefunden werden könnten. Bleibt die Störung dennoch bestehen, wird die Zentrale dies dem Agenten via chiffrierter Standard-Nachricht mitteilen, aber keine „geheimen“ Nachrichteninhalte mehr an den Agenten übermitteln, da eine Kompromittierung des Agenten nicht ausgeschlossen werden kann. Der Agent hat daraufhin die Möglichkeit, seine sendeseitigen Schlüssel (ggf. nebst Überladedateien) zu überprüfen. Kann er (ggf. trotz Wiederholung) die Situation nicht ändern, wird er die Kommunikation mit der Zentralen einstellen und sich als „kompromittiert“ betrachten.

Es sei nochmals betont, dass hier davon ausgegangen worden ist, dass der Gegner

- **alle genannten (auch internen) Regelungen sowohl bzgl. der Agenten als auch bzgl. der Zentrale** genauestens kennt,
- **den Algorithmus in allen Details** kennt und
- in der Lage ist, **den gesamten Kommunikationsverkehr zwischen Agenten und Zentrale** lückenlos abzuhören und aufzuzeichnen.

Dennoch ist der Gegner mit der hier beschriebenen Vorgehensweise nur punktuell in der Lage, an wenige geheime Informationen zu gelangen, wenn die Agenten entsprechende Aufmerksamkeit walten lassen, und das ohne irgend eine Änderung von Schlüsseln.

Wie die Agenten sicher stellen, dass ihr jeweiliger Standort nicht ermittelt werden kann und weiteres mehr, ist nicht Gegenstand dieser Untersuchung. Da man via Handy usw. ins Internet gelangen kann, gibt es hier sicherlich zunächst keine wesentlichen Einschränkungen und damit Probleme.

Vielmehr sollte hier an vorstellbaren Situationen erörtert werden, welche konkreten Möglichkeiten sich mit CODING in der Praxis ergeben. Natürlich ist es auch möglich, online-Verbindungen (z.B. via „Voice over IP“) mit dem Programm CODING zu sichern bzw. Handy-Hardware zur Sicherung zu verwenden. Die genauen Modalitäten müssten hierfür jedoch genauer erörtert werden.

\- - -