CODING - Stream Cipher Methods by Varying Components during Ciphering Data

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

Abstract

Kernel of the symmetric block ciphering methods presented here is the coupling of XOR operations and of invertible substitution tables **S** with all possible 256**t byte groups (with t=1, 2, 3, ... Bytes, fixed at the beginning) being derived from keys:

K(block) := S(S(block) ⊗ E_o) ⊗ E_u with

- E_o upper and E_u lower triangular (byte-group-)matrix with (byte-block-length/t)**2 values, value 1 at all non-zero positions,
- ⊕ the byte-group-wise addition without carry ('xor'; 'not xor' is possible too),
- ⊗ the (vector) multiplication which belongs to ⊕.

Variable block lengths (v*t or (mod t)>0) are possible. This kernel can be applied n-times:

$\mathbf{K_n}$(block) := K(...K(block)...) with n K-operations, in which n can be variable.

Because XOR operations and S-tables only operate in a useful manner if 'block' is not to "homogeneous" and for safety, two further components are determined from keys

- parameters of 2 pseudo random processes, - operation key

used at beginning and at end to get a ciphered block:

cblock := S(ZZ_2 ⊕ S(Op_E ⊕ S(K_n(Op_A ⊕ S(ZZ_1 ⊕ S(block)))))) with

- ZZ_1 and ZZ_2 are the bytes of the 1. and 2. pseudo random number process in block length,
- Op_A and Op_E is the (1./front and 2./back part of the or multiple of the) operation key.

An initial key is first expanded to t*256**t bytes (all further keys have this size too) and can be modified so the result key does not statistically differ from a random key.

Using an invertible S-table, the value (modulo n) of only as much consecutive bits of a key as to represent the number n-1 is determined to shift the last n S-table elements cyclically in accordance with this value, n=2 to 256**t. So all such 256**t! tables can be generated by the top bits of all possible keys and have length of t*256**t bytes.

The byte-group-value +1 at a position of a S-table determines the byte-group in the key from which up 2*7 bytes are used to initialize two floating point numbers (IEEE 754) for a pseudo random process. Floating point numbers are initialized again if a process will be cyclic.

Idea is, to modify (operation) keys similar to data blocks to generate and use more or less continual new S-tables, new pseudo random processes, and new operation keys during ciphering data.

Inspections show that in spite of knowledge of 2 of the 3 components S-table, pseudo random parameters, and operation key as well as the knowledge of original and ciphered data it can not infer the missing 3. component if component modifications are carried out "some time".

As well it is shown that by knowledge of the 3 components generated by a key the key itself can not be inferred (because of usage of interim operation keys). That is compromising of data and with that of components does not concern data ciphered before component-changing to the compromised components. By add-on usage of separate components only for the modifications of keys, it will be guaranteed that data sections ciphered after a component-changing started from compromised components are not compromised automatically.

Because of that a safety stream ciphering should be possible as already constructed for t=1,2,3.

Introduction / Summary

Kernel Definition

- Kernel of the ciphering methods presented here is the coupling of XOR operations and of invertible substitution tables **S** for **byte-groups** with **t**=1,2,3,… bytes (fixed at the beginning):

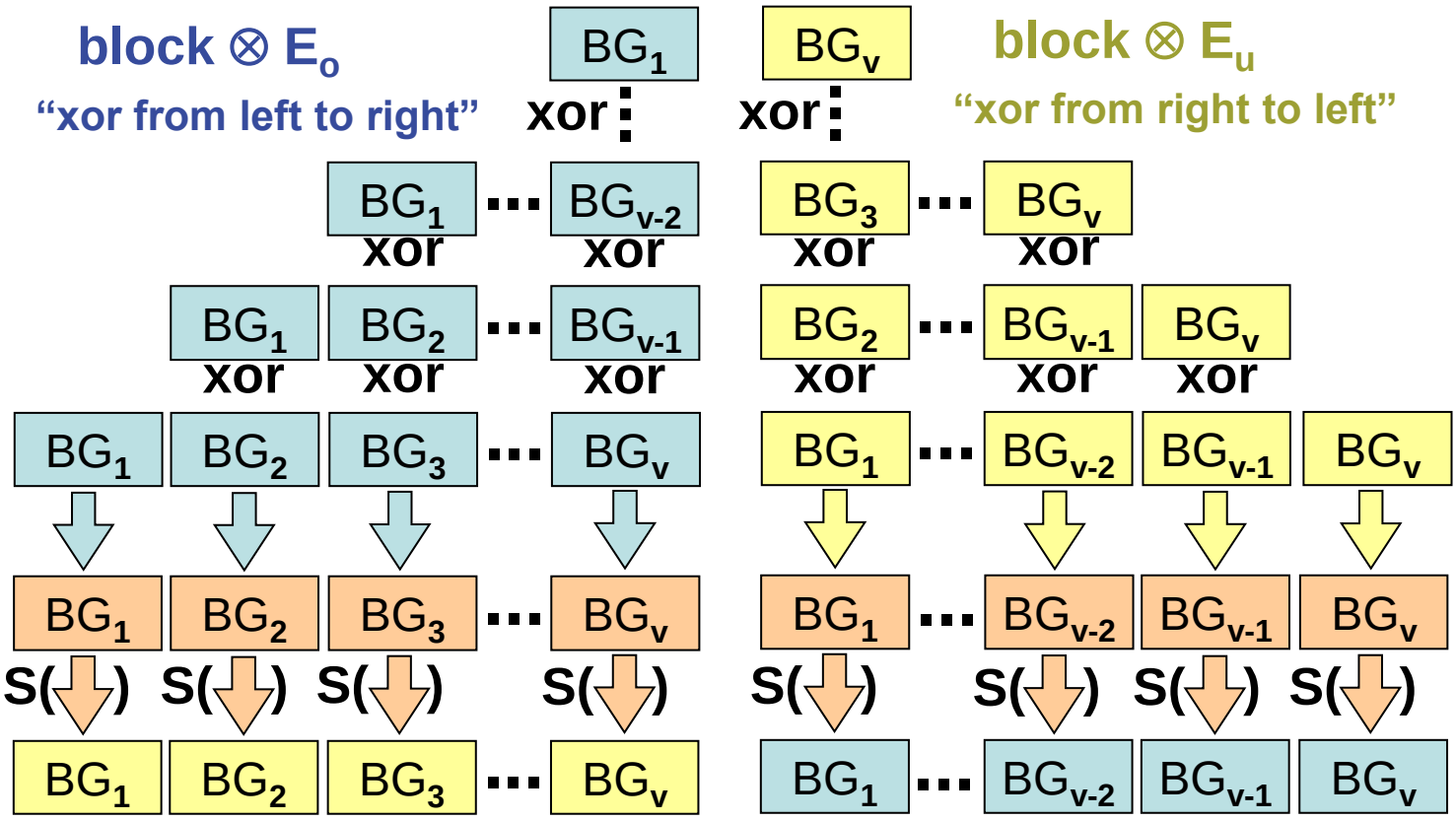
  $$K(block) := S(S(block) \otimes E_o) \otimes E_u \quad \text{with}$$

  - E_o upper and E_u lower triangular (byte-group-) matrix with (byte-block-length/t) ^{2} values, value 1 at all non-zero positions,
  - $\oplus$ the byte-group -wise addition without carry (‘xor'; 'not xor' is possible too),
  - $\otimes$ the (vector) multiplication which belongs to $\oplus$.
- This kernel can be applied **n-times**:

  $$K_n(block) := K(...K(block)...) \quad \text{with}$$

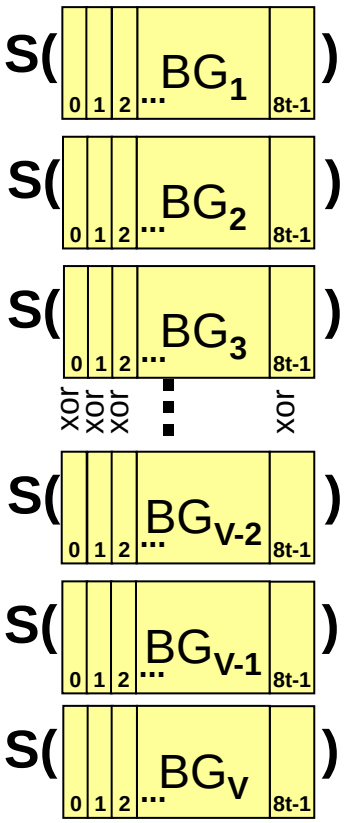
  - n K-operations, in which n can be variable.
- Blocks of variable-(byte-)length v*t can be used, the last block with length modulo t >0 too.

Kernel Explanation (1)



BG_x = byte-group x
v = number of byte-groups in block
S(.) = S-tables application

Kernel Explanation (2)



- The XOR operation connects the bits in columns about all byte-groups separated by each bit position.
- The application of the substitution table S modifies and “connects” the bits in lines per byte-group.
- The substitution table S is used between the XOR steps too, otherwise ”xor from right to left” only has the effect of a beginning permutation before “xor from left to right”.
- These operations can be seen as a form of a “bit shredder” in which all bits of a block interacts with each other. The combination guarantees the real influence of each bit of a block to each other bit of such block.
- As shown in the „Piling-up-Lemma“, XOR-connected stochastic independent binary random variables converge against the uniform distribution of 0 and 1.
- If you connect n data sets each of them with the same binary distribution p being independent one another by XOR operation, the following is valid for the distribution $\mathbf{p_G}$ of the result set **G**:
  $|(p_G-(1-p_G))| = |(p-(1-p))^n| \Rightarrow p_G \in \{½ \pm ½^* |(p-(1-p))^n|\}$.
- Example: n=512, p=0.01 $\Rightarrow |p_G-½| \approx 1.61*10^{-5}$
  p=0.001 $\Rightarrow |p_G-½| \approx 0.1794$.

Definition of the Cipher Envelope

- Because XOR operations and S-tables only operate in a “useful” manner if 'block' is not to "homogeneous" and for safety, two further components are determined from keys
  - parameters of 2 pseudo random processes,
  - operation key,

  used at the beginning and at end to get a ciphered block:

  cblock := $S(ZZ_2 \oplus S(Op_E \oplus S(K_n(Op_A \oplus S(ZZ_1 \oplus S(block))))))$ with
  - **ZZ_1** and **ZZ_2** are the bytes of the 1. and 2. pseudo random number process in block length,
  - **Op_A** and **Op_E** is the (1./front and 2./back part of the or multiple of the) operation key.

Keys (1)

- An user **key** is first expanded to **t*256^t bytes** of the initial key by copying, all further keys have this size on principle.
- Because this is not a “useful” solution, a method is used which is called **key overloading** / **overlaying**:
  - Choose datasets forming one set of data.
  - Declare, how frequently this set of data has to be ciphered.

  Now **t bytes** of the ciphered set of data are interpreted as **position value** in key and further **t bytes** are interpreted as **key value** each and the initial key is changed accordingly.
- It can be shown (see “Convergence of Overlay Operations at Keys” later) that a **random key** with 256^t byte-groups has for t=**1: 61.91%,** t=**2: 61.80%,** t=**3: 61.80%** of all possible byte-groups in average [**expected value E(X,t)**].
- By **empirically** explored overloadings using ciphering methods presented here it is: t=1: 63.28%, t=2: 63.22%, t=3: 63.21%.

Keys (2)

- The **Golomb Axioms** to assessment of pseudo random processes are ”static” and pseudo random processes provably fulfilling these axioms are “predictable” instead of ”random” in most cases.
  I.e. the proof of these axioms only results in “fulfilled” or „not fulfilled“ or “not determined”.
- But if it is build (position,value)-tuples out of a pseudo random process and an “homogeneous” key of 256^t byte-groups is overloaded by these tuples, then it can be evaluated how much percent of all possible byte-groups the overloaded key covers in average.
- The nearness to the theoretical expected values of such evaluated values can be interpreted as **quality measurement for random**.
- The number of overloadings is not suitable, because here first a significance level has to be fixed inter alia to get a comparison criterion (as per inspections in “Convergence of Overlay Operations at Keys” later) etc.

Substitution Tables S

- An **invertible substitution table S** is a table with exact all 256^t possible byte-groups as table elements.
- All invertible S-tables can be generated by the possible values of the **first (8t–1)*256^t+1 key bits**:
  - S-table being initialized with all byte-groups as ascending values (identical function) or an arbitrary invertible S-table.
  - The value (modulo n) of only as much consecutive bits of a key as to represent the number n-1 is determined to shift the last n S-table elements cyclically in accordance with this value, n=2 to 256^t.
- *t*: *number of S-tables* $256^t!$: *key size* $t*256^t$*:* *key space* $(256^t)**(256^t)$:

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 Random Processes

- A key and a S-table can be used to generate the parameters for one or more **pseudo random number processes**.
- In the ciphering methods presented here the byte-group-value +1 at a position of a S-table determines the byte-group in the key from which up 2*7 bytes are used to initialize two floating point numbers (according to IEEE 754) for a pseudo random process.
- The mantissa bytes of the result of the multiplication of the squared 1. with the squared 2. floating point value give “pseudo random bytes", whereas the squared 1. is set as new 2. floating point number and both characteristics are set to zero etc.
- Floating point numbers are initialized again if a process will be cyclic.
- The random bytes are responsible for the **“noise”** because in general the data has not to be dissimilar. Only the dissimilarity makes it possible that S-tables work in wide range of sections and therefore has the maximum effect as it has for XOR operations.

Change of Components (1)

- A **change of components** is defined as the transition from initial key or from a **component record** consisting of
  - (inverse and) S-table,
  - 2 pseudo random number processes,
  - operation key

  to the first or next such component record.
- Initial point is the **(operation) key** which is treated **as data block** in this connection.
- **Generating components** the following steps are carried out:
  - **A**. generate **1. interim operation key**
  - **B**. generate **(possibly inverse and) S-table**
  - **C**. generate **2. interim operation key**
  - **D**. initialization of the **2 pseudo random number processes**
  - **E**. generate **real / initial operation key**
- in which **operation keys** are generated as follows (e.g.):
  - **(1)** 'xor' random numbers 1 (ZZ_1; key)
  - **(2)** process S-table (B for C, D and E; predecessor of B for A)
  - **(3)** 'xor' of key data from left to right
  - **(4)** process S-table (B for C, D and E; predecessor of B for A)
  - **(5)** 'xor' of key data from right to left
  - **(6)** process S-table (B for C, D and E; predecessor of B for A)
  - **(7)** cyclical shift (value of result (4) and current random number)
  - **(8)** repetition of the steps (3) to (7)
  - **(9)** 'xor' random numbers 2 (ZZ_2; key)
  - **(10)** process S-table (B for C, D and E; predecessor of B for A)
  - **(11)** repetition of steps (3) to (8), step (3) and (5) transposed.
- Using such change of components, by knowledge of the 3 components generated by a key the **key** itself **can not be inferred** (see inspections in "Examination of the Safety of the Algorithm in Brief" later).

Change of Components (2)

- In spite of knowledge of 2 of the 3 components as well as the knowledge of original and ciphered data it can **not infer the missing 3. component** (see inspections in “Examination of the Safety of the Algorithm in Brief” later).
- Among other things this can be ensured because of that
  - the operation key is used a maximum of 256^t -1 times before being changed by
    $$\textbf{operation key}_{new} := K_n(\textbf{operation key}_{old}),$$
  - floating point numbers are initialized again, if a pseudo random number process will be cyclic,

  if a **total change of components** doesn’t happen before which is initialized by the repeat performance of the 1. pseudo random number process at the latest (or after treatment of a fixable average number of data blocks).
- That is compromising of a component record and with that of data ciphered by these components does **not concern data** ciphered **before component-changing** to the compromised component record.
- By add-on usage of **separate components only for the modifications of keys** before the original change of components, it will be guaranteed that **data** sections ciphered **after a component-changing** started from compromised components and thereby used components are **not compromised** automatically.
- Therefore a **safety stream ciphering** of arbitrary extensive data over a very long period of time without additional exchange of keys or components should be possible.

Size of Parameter Space

- In common there is for the **size of CODING parameter space**
  npars := **nstabs * nkeys * nrannums / delta**
  ca.= 10**1 155.40 | 10**602 877.14 | 10**235 135 154.14 for t=1 | 2 | 3
  with
  - **nstabs** := number of S-tables (→ see page 5),
  - **nkeys** := size of key space (→ see page 5),
  - **nrannums** := size of random number space
    = 2**(14*8) ca.= 10**33.71,
  - **delta** := **nkeys** / **unikeys**
    ca.= 10**1.75 | 10**3.11 | 10**4.40 for t=1 | 2 | 3,
  - **unikeys** := number of binary uniform distributed keys (motivated by ‚xor')
    = (**keylen***8)! / (**keylen***4)!^{2},
  - **keylen** := key-length in bytes (→ see page 5).

Group Membership

- It still has to be settled, to which minor groups of the **symmetric group S(npars)** the groups of elements of the parameter spaces of each CODING method are isomorphic. I.e. which **finite simple groups** can generate these parameter groups.

Key-Length

- Statements about maximum key-lengths are things of the past:
  - The usable CODING methods are working with **initial key-length up to 50 MB** (exactly 48*1024^2 bytes).
  - Using the CODING method, you can generate processes which can handle **any initial key-length** (even if it is less "practicable" for t>3 at the moment).

Guaranteed Future

- The **„rabbit and hedgehog"-play** (Brothers Grimm) between encipherers and crypto analysts is **finished**, if the cycle lengths of the parameters of the CODING methods in connection with (pseudo) random initial keys – possibly after method modifications – are turned out to be „sufficient long".
  Beside changing parameters you can change **from** CODING processes of **t to** such of **t+1** if t seams to be safe enough no longer.

Hardware-Box

- The stream ciphering (for example) could be managed by **2 hardware boxes**:
  - **Assembled** they first generate a **real random key** – by cosmic radiation for example.
  - **After that they are separated** and moved to the places where the data stream should be en- and deciphered.
  - Using the today „most protected“ CODING method (t=3) the **data stream** between the boxes should **not** be **plain-text readable** for any organization with any future infrastructure **for the foreseeable future**.

Efficiency

- Today the middle CODING method (**t=2**) can cipher with about **20 MB/sec throughput rate** on a 2-processor PC (x64 system, Intel i7 CPU, 2.67MHz, 8192MB core), for **t=3** the efficiency is far smaller at the moment and possibly has to be **optimized** before application.

Convergence of Overlay Operations at Keys

Initial Position

By overlaying keys there quickly is the question, how many overlay events have to occur to be unable to tell the result key modified in such way from a "random" key. But what is a "random" key?

A key has **S positions** in the **position set *S*** and **W different values** in the **value set *W***, W≤S and **w(s)** is the **value w** at the **position s** in the key. Be **Z a random process** with **tuples T**:=**(s(Z),w(Z))** of all possible values and positions with single values **w(Z)** and positions **s(Z)**, in which w(Z), s(Z) are independent of each other. Be **W=n** the number of elements of *W* and **#(w)**, w∈*W* the number of the same value w in accordance with all positions *S*.

With any tuple (s(Z),w(Z)) the value w at the position s(Z) in the key is replaced by the value w(Z): **w(s(Z))** := w(Z) . Correspondingly $\boldsymbol{W_{old}}$ and $\boldsymbol{W_{new}}$ denote the set *W* before and after the replacement and $\mathbf{W_{old}}$ and $\mathbf{W_{new}}$ denote the different numbers of values at these moments.

1. Problem: About which number W=N of different values the key fluctuates using a random process Z ?

It is:

$P(\text{a position s(Z) from Z}) = 1/S,$
$P(\text{a value w(Z) from Z}) = 1/S,$
$P(w(Z)\in W) = P(w(Z)=w_1\in W) + P(w(Z)=w_2\in W) + ... + P(w(Z)=w_n\in W) = n/S,$
$P(w(Z)\notin W) = (S-n)/S$.

If $w(Z)\notin W$, then it is:

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

If $w(Z)\in W$, then it is:

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

That is why the following values are determined:

$P(W_{new} = W_{old}),$
$P(W_{new} = W_{old} + 1),$
$P(W_{new} = W_{old} - 1)$.

So be **D** the **number of multiple values *D*** of *W*, i.e. $w\in D \Rightarrow \#(w)>1$, D≤W and $\mathbf{S_D}$ the **number of positions $\boldsymbol{S_D}$ with multiple values** *D*. Then it is:

$S-n \le S_D \le 2*(S-n)$, if $n \ge S/2$
$\le S,$ else.

$S-S_D$ is the number of positions with values $w\in W$, #(w)=1, i.e. of course $w\in W-D$, therefore it is:

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

On the assumption of a fixed n and S_D value (n,S_D), it follows from that:

$P(W_{new}=W_{old} \text{ for } (n=W_{old},S_D))$
$= P(w(Z)\notin W \wedge w(s(Z))=w \wedge \#(w)=1)$
$+ P(w(Z)\in W \wedge w(s(Z))=w \wedge \#(w)\neq 1)$
$+ P(w(Z)\in W \wedge w(s(Z))=w \wedge \#(w)=1 \wedge w(Z)=w(s(Z)))$

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

If $n=S$ then it is $S_D=0$ and therefore $\Rightarrow =1/S$ (✓).
If $S_D=S \Rightarrow$ every value is available multiple $\Rightarrow =n/S$ (✓).

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

If $n=S \Rightarrow =0$ (✓).
If $S_D=S \Rightarrow$ every value is available multiple $\Rightarrow =1-n/S$ (✓).

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

If $n=S$ then it is $S_D=0$ and therefore $\Rightarrow =1-1/S$ (✓).
If $S_D=S \Rightarrow =0$ (✓).

and

$P(W_{new}=W_{old} \text{ for } (n,S_D)) + P(W_{new}=W_{old}+1 \text{ for } (n,S_D)) + P(W_{new}=W_{old}-1 \text{ for } (n,S_D))$
$= 1 -(n-1)/S + S_D*(2*n-S-1)/S^2 + S_D*(S-n)/S^2 + (n-1)*(S-S_D)/S^2$
$= [S^2 -n*S+S +2*n*S_D-S_D*S-S_D+S_D*S-n*S_D +n*S-n*S_D-S+S_D] /S^2$
$= [S^2+(nS-nS)+(2*n*S_D-n*S_D-n*S_D)+(S_D*S-S_D*S)+(S-S)+(S_D-S_D)] /S^2 =1$ (✓)

Building up:

$P(W_{new}=W_{old}+1 \text{ for } S>n\geq S/2)$
$= \sum_{s\in S_D} P(W_{new}=W_{old}+1 \text{ for } (n,s))$

→ Like this the weighting in accordance with the number of possible S_D variants is missing !! ←

There is (using [(1)]):
Be $S_D=S-n+k$: **Weighting Gw(S_D)** := $\binom{S-n-1}{k-1} * \binom{n}{k}$ with $\binom{S-n-1}{k-1}=1$ for k=1.

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

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

It follows from that:

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

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

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

See also [(6)] below for the test of correctness of these values.

Simple assumption of balance:

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

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

It can be verified by the following formulation too that this assumption results in success:

Expanded system:

It is searched for the value N with the maximum probability: $N := \max_{n=1,...,S}[P(W=n)]$.

During this the P(W=n) are calculated with the following system of equations:

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

The first n equations aren't independent, so that one of these remains unconsidered and is replaced by the last equation.

For the exemplary and simple calculations see [(8)] for S=2 and see [(9)] for S=3.

In general it is for all n=1,...,S:

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

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

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

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

Then it is [to the formulation by determinants see [10]]:

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

or

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

brief $C * x = b$.

From that it is possible to generate an upper triangular matrix which n∗(n+1)/2 elements are known to be transformed via index transformation

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

in an 1-dimensional array.

On closer examination, however, there is the following system:

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

in which

$\underline{r}_1 = \underline{m}_2 = \underline{a}_1 = \underline{b}_1 = 1$,

$\underline{r}_i = 0 - \underline{r}_{n-1}*p_{i-1}/\underline{a}_{i-1}$, 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,

and, to complete, be

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

Therefore only the 4–tupel values

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

have to be stored and modified to hold the values of the triangle matrix, the input values p_i, a_i, m_i, and b_i are just calculable and determinable in a direct way.

With that the x_i for i=n to 1 can be calculated as follows:

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

only for $i<n-1$ ◄ ► only for $i<n$

or

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

If you calculate the values N with a special program you can find

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

i.e. the method of the simple assumption of balance is completely confirmed.

The values of the concrete cases checked by the author laid near approx. 63,28% and 63,22% and 63,21% (see (14), (15), and (16) below) respectively. Possibly these differences to the theoretical values calculated here should be examined more exactly.

2. Problem: How many tuples have to be generated via random number process Z and have to apply to the key until P(W=N) is reached surely ?

- There have to be at least $N-N_0$ tuples if $\mathbf{N_0}$ is the **number of different bytes (byte-groups) already included in the original key** and $N_0<N$.
- The term „is reached surely“ is taken more precisely [because P(W=N) has to be smaller than 1 for $N_0<N$] e.g. with the aid of a **significance level s** of e.g. s=95%.

If you define

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

there is in general for n>N

$P(W_{new}=W_{old}=n) := 0,$
$P(W_{new}=W_{old}-1,W_{old}=n) := 0,$
$P(W_{new}=W_{old}+1,W_{old}=n) := 0$

and the total system remains in status N if reached first.

If the following P terms are the probabilities

$P_i(n)$ to be in status W=i after the nth tuple replacement,
$P_{i-1,i}$ to change from status W=i−1 to status W=i by a tuple replacement, [see p_{i-1} above],
$P_{i,i-1}$ to change from status W=i to status W=i−1 by a tuple replacement, [see m_i above],
$P_{i,i}$ to stay in status W=i by a tuple replacement [**á$_i$** := a_i+1 see above],

Then there is

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

or with

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

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

there is

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

As an example for N=3 see [11].

So if P(0) with $i=N_0$ has a 1 at the ith position i.e. $P_i(0)=1$ and $b_{N,i}(n)$ is the value in Nth row and ith column of the matrix $(B)^n$, then it is searched for the smallest value n resulting in: $b_{N,i}(n) \geq s$.

Then #(T)=n is the searched tuple value in connection with significance level s.

The matrix P shown here is a stochastic matrix i.e. it is a transition matrix of a (finite) Markoff chain with absorbed status N [in which rows and columns are exchanged here].

On closer examination it is

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

In general it can be determined for any n:

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

It is for special n<N (see [12] too):

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

Because P(n) can be described by $P(n) = (B)^n * P(0)$ and only the vales $b_{N,i}(n)$ with a fixed i are interested, the interest in the matrix $(B)^n$ can be reduced to the interest of its last (N.) row.

Additionally it arises (see [13]) that $b_{N,i} < b_{N,k}$ if $i<k$ and $b_{N,k} \neq 0$. If that is why the $b_{N,1}(n)$ are looked at, the $b_{N,i}(n)$ with $i>1$ can at most be bigger than or are in the near bounds of $b_{N,1}(n)$ [=:**$P(1 \rightarrow N|n)$**] supposed a big N and "small" distances to 1.

If you calculate the values N with a special program you can find the smallest n with $P(1 \rightarrow N|n) > s$ as

(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	?
empirical determined: [see [14], [15], [16] above]	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)

In which

nV := number of (single) overlay events looked at,
nV* := number of overlay events looked at where statistics are ascertained only all 1066 overlay events to calculate the above values,
mZH := min(max(frequency of byte-groups)) with n, last mZH value to nV or nV*,
(N=...) := value N calculated from nV or nV*.
...(?) assessed value; the program internal value representation (mantissa precision) as well as the computer speed (PC of the author) is not sufficient for S=16777216 to get an result in acceptable time even for s=1% (!).

Explanations:

For simplification reasons the terms $\binom{n}{k}$ are called „**(n over k)**“ in the following too.

(1) S_D=S−n+1: Exactly one value is repeated and there are n values possible:
Weighting =n.

S_D=S−n+2: Exactly 2 values are repeated and there are n(n−1) values possible from which the first is repeated x_1=2 to x_1=S−n times and the 2. x_2=S−n+2−x_1 times respectively. Because it doesn't matter which one is the 1. and which one is the 2. value, it holds:
Weighting =n!/(2!(n−2)!)(S−n−1).

S_D=S−n+3: Exactly 3 values are repeated and there are n(n−1)(n−2) values possible from which the 1. is repeated x_1=2 to x_1=S−n−1 (=S−n+3−2∗2) times, the 2. x_2=2 to x_2=S−n+1−x_1 (=S−n+3−1∗2−x_1) times, the 3. x_3=S−n+3−x_1−x_2 times respectively:
Weighting =n!/(3!(n−3)!)∗[(S−n−2)+(S−n−3)+...+1]
=(n over 3)(S−n−2)((S−n−2)+1)/2.
=(n over 3)((S−n−2)+1)∗(S−n−2)/(1∗2).

S_D=S−n+4: Exactly 4 values are repeated and there are n(n−1)(n−2)(n−3) values possible from which the 1. is repeated x_1=2 to x_1=S−n−2 (=S−n+4−3∗2) times, the 2. x_2=2 to x_2=S−n−x_1 (=S−n+4−2∗2−x_1) times, the 3. x_3=2 to x_3=S−n+2−x_1−x_2 (=S−n+4−1∗2−x_1−x_2) times, the 4. x_4=S−n+4−x_1−x_2−x_3 times respectively:
Weighting =n!/(4!(n−4)!)∗[(S−n−3)(S−n−2)/2+(S−n−4)(S−n−3)/2+...+1∗2/2]
with $1*2+2*3+3*4+...+n*(n+1) = 1/3*n^3+n^2+2/3*n$
=(n over 4)[$(S-n-3)^2$+3(S−n−3)+2]∗(S−n−3)/(2∗3).
=(n over 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]:
Exactly S−n values are repeated and there are n(n−1)...(n−(S−n−1)) values possible and all values precisely exist 2 times. I.e.
Weighting =(n over S−n).
Check: $\prod_{i=0,...,(S-n-2)}[(S-n-(S-n-1))+i]$ ∗(n over S−n)/(S−n−1)!
= (S−n−1)! ∗(n over S−n)/(S−n−1)! = (n over S−n).

S_D=S−n+n=S [n<S/2]:
Exactly n values are repeated and are possible from which the 1. is repeated x_1=2 to x_1=S−2n+2 (=S−n+n−(n−1)∗2) times, the 2. x_2=2 to x_2=S−2n+4−x_1 (=S−n+n−(n−2)∗2−x_1) times, ... , the (n−1). x_{n-1}=2 to x_{n-1}=S−2−$\sum_{i=1,...,(n-2)}[x_i]$ (=S−n+n−1∗2−$\sum_{i=1,...,(n-2)}[x_i]$) times, the n. x_n=S−$\sum_{i=1,...,(n-1)}[x_i]$ respectively.
Weighting =(S−n−1 over n−1).

(2) $\sum_{j=1,...,(S-n)}$[(S−n−1 over j−1)∗(n over j)] ⇒ according to Knuth,vol.I,p.53,(7)
= $\sum_{j=1,...,(S-n)}$[j/(S−n) ∗(S−n over j)∗(n over j)]
= 1/(S−n)∗$\sum_{j=1,...,(S-n)}$[j∗(S−n over j)∗(n over j)] ⇒ according to Knuth,vol.I,p.59,(Prob.1)
= 1/(S−n)∗(S−n+n−1 over n−1)∗(S−n)
= (S−1 over n−1)

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

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

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

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

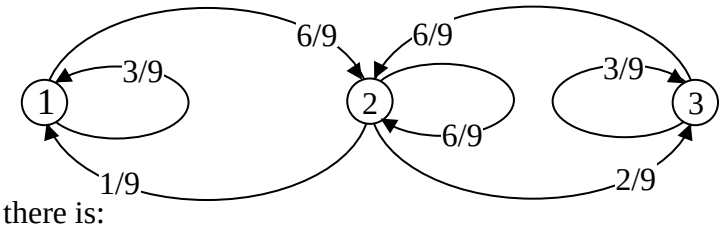
(5) $\sum_{S-n<s\le 2*(S-n)}[(1-(n-1)/S + s*(2n-S-1)/S^2) * Gw(s)]/Norm$
$= \sum_{k=1,...,(S-n)}[(1-(n-1)/S + (S-n+k)*(2n-S-1)/S^2)*(S-n-1 \text{ over } k-1)*(n \text{ over } k)]/Norm \Rightarrow$
with: $S^2 - S*(n-1) + (S-n+k)*(2n-S-1)$
$= S^2 - S*n + S + S*(2n-S-1) - n*(2n-S-1) + k*(2n-S-1)$
$= S^2 - S*n + S + 2S*n - S^2 - S - 2n^2 + S*n + n + k*(2n-S-1)$
$= 2S*n - 2n^2 + n + k*(2n-S-1)$
$= 2n*(S-n+1/2) + k*(2n-S-1)$
$= 2n*(S-n+1/2)/S^2 * \sum_{k=1,...,(S-n)}[(S-n-1 \text{ over } k-1)*(n \text{ over } k)]/Norm$
$+(2n-S-1)/S^2 * \sum_{k=1,...,(S-n)}[k*(S-n-1 \text{ over } k-1)*(n \text{ over } k)]/Norm \Rightarrow$ (2) and (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) $P(W_{new}=W_{old}+1 \text{ for } S>n\geq S/2) + P(W_{new}=W_{old}-1 \text{ for } S>n\geq S/2) + P(W_{new}=W_{old} \text{ for } S>n\geq S/2)$
$= (S-n)^2*(S+n-1)/[S^2*(S-1)] + n*(n-1)^2/[S^2*(S-1)] + [n*(S-n+1)*(S-1) +2n*(n-1)*(S-n)] /[S^2*(S-1)]$
$= [(S-n)^2*(S+n-1) + n*(n-1)^2 + n*(S-n+1)*(S-1) + 2n*(n-1)*(S-n)] /[S^2*(S-1)]$
$= [(S^2-2nS+n^2)*(S+n-1) + (n^2-2n+1)*n +(S^2-nS+S-S+n-1)*n + 2n*(nS-n^2-S+n)] /[S^2*(S-1)]$
$= [S^3-2nS^2+n^2S+nS^2-2n^2S+n^3-S^2+2nS-n^2+n^3-2n^2+n+nS^2-n^2S +n^2-n +2n^2S-2n^3-2nS+2n^2] /[S^2*(S-1)]$
$= [S^3-S^2+n*(-2S^2+S^2+2S+1+S^2-1-2S)+n^2*(S-2S-1-2-S+1+2S+2) +n^3*(1+1-2)] /[S^2*(S-1)]$
$= 1$ (✓).

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

(8) S=2: With
$P(W_{new}=W_{old}+1) = (n+1)*(2-n)^2/4 \Rightarrow$ n=1: 1/2; n=2: 0,
$P(W_{new}=W_{old}-1) = n*(n-1)^2/4 \Rightarrow$ n=1: 0; n=2: 1/2,
$P(W_{new}=W_{old}) = n*(2-n+1)/4 \Rightarrow$ n=1: 1/2; n=2: 1/2
there is:
$P(W=1) = P(W=1)*P(W_{new}=W_{old},W=1) + P(W=2)*P(W_{new}=W_{old}-1,W=2)$
$P(W=2) = P(W=1)*P(W_{new}=W_{old},W=2) + P(W=2)*P(W_{new}=W_{old}+1,W=1)$
$P(W=1)+P(W=2) = 1$
$\Rightarrow P(W=1) = P(W=1)*1/2 + P(W=2)*1/2,$
$P(W=2) = P(W=1)*1/2 + P(W=2)*1/2,$
$P(W=1)+P(W=2) = 1$
$\Rightarrow P(W=1) = P(W=2),\ 2*P(W=1) = 1$
$\Rightarrow P(W=1) = 1/2$ and $P(W=2) = 1/2$.

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



there is:
$P(W=1) = P(W=1)*P(W_{new}=W_{old},W=1) + P(W=2)*P(W_{new}=W_{old}-1,W=2),$
$P(W=2) = P(W=2)*P(W_{new}=W_{old},W=2) + P(W=1)*P(W_{new}=W_{old}+1,W=1)$
$+ P(W=3)*P(W_{new}=W_{old}-1,W=3),$
$P(W=3) = P(W=3)*P(W_{new}=W_{old},W=3) + P(W=2)*P(W_{new}=W_{old}+1,W=2),$
$P(W=1)+P(W=2)+P(W=3) = 1$
$\Rightarrow P(W=1) = P(W=1)*3/9 + P(W=2)*1/9,$
$P(W=2) = P(W=2)*6/9 + P(W=1)*6/9 + P(W=3)*6/9,$
$P(W=3) = P(W=3)*3/9 + P(W=2)*2/9,$

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 and P(W=3) = 2/9
$\Rightarrow$ N = 2 with P(W=N) = 6/9.

(10) With the determinants

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

and $\mathbf{D_i}$:= {column i in D exchanged by b} for all i=1,...,S
P(W=i) = D_i/D results in the solution of the equation system.

[Because the factor f := $1/(S^2*(S-1))^{(n-1)}$ can be moved in front of D and D_i (each n−1 corresponding rows), the values of D_i/D would not change if you use the following values instead of the previous values a_n, p_n, and m_n:
$a_n = n*(S-n+1)*(S-1) + 2n*(n-1)*(S-n) - S^2*(S-1)$,
$p_n = (S+n-1)*(S-n)^2$,
$m_n = n*(n-1)^2$.]

Further there is:

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

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

...

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

Be

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

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

Then there is for k=2,...,n−1:

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

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

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

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

$D = \sum_{k=3,...,n+1}[q_{3,k}*A_k] \Rightarrow$ in general with 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.

Using D_i, i=1,...,n there is correspondingly:

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

$\Rightarrow$ in general with j=i+2,...,n and i<n:

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

The representation for D and D_i chosen here is enabling a relative simple calculation of these values using a special program.

(11) Given the relationships of (9) above. Then there is:

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

and with N_0=1

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

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

(12) For j=1,...,N−1 and i=1,...,N there is

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

$b_{i,j}(3) = \sum_{k=1,...,N}[b_{i,k}(2)*b_{k,j}(1)] = b_{i,j-1}(2)*b_{j-1,j}(1) + b_{i,j}(2)*b_{j,j}(1) + b_{i,j+1}(2)*b_{j+1,j}(1)$
$= b_{i,j-1}(2)*m_j + b_{i,j}(2)*á_j + b_{i,j+1}(2)*p_j$

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

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

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

with **z**:= $3+(2k+1)*(2j-r-t)+2j*(r+t-2j)$ $[=(3+r+t-2j)*(j-k)+(3-r-t+2j)*(k-j+1)]$

$$b_{i,i+n}(n) = \prod_{k=i+1,...,i+n}[m_k] \quad \text{for all } i+n<N, \Rightarrow \text{with: } q := [S^2*(S-1)]^n$$
$$= (i+1)*i^2 * (i+2)*(i+1)^2 * (i+3)*(i+2)^2* ... * (i+n)*(i+n-1)^2 /q$$
$$= (i+n)!/i!*(i+n-1)!/(i-1)!*(i+n-1)!/(i-1)!/q$$
$$= (n+i \text{ over } i)*[n!*(n+i-1 \text{ over } i-1)]^2*n!/q$$
$$= (n+i \text{ over } i)*[(n+i-1 \text{ over } i-1)]^2*(n!)^3/q$$
$$= (n+i \text{ over } i)*[(i/(n+i))*(n+i \text{ over } i)]^2*(n!)^3/q$$
$$= (i/(n+i))^2 *[(n+i \text{ over } i)*n!]^3/q$$
$$= (i/(n+i))^2 *[(i+n)!/i!]^3 / [S^2*(S-1)]^n$$

$$b_{i+n,i}(n) = \prod_{k=i,...,i+n-1}[p_k] \text{ for all } i+n\le N,$$
$$=(S+i-1)*(S-i)^2*(S+i)*(S-(i+1))^2*(S+i+1)*(S-(i+2))^2*...*(S+i+n-2)*(S-(i+n-1))^2/[S^2*(S-1)]^n$$
$$= (S+i+n-2)!/(S+i-2)!*(S-i)!/(S-i-n)!*(S-i)!/(S-i-n)!/q$$
$$= (S+i+n-2 \text{ over } S+i-2)*n!*[(S-i \text{ over } S-i-n)*n!]^2/q$$
$$= (S+i+n-2 \text{ over } n)*(S-i \text{ over } n)^2*(n!)^3/[S^2*(S-1)]^n$$

$$b_{i,j}(n) = \prod_{k=i+1,...,j}[m_k]*\sum_{k=i,...,j}[á_k], \quad \text{if } j=i+n-1 \text{ for all } 1<n<N-i-1,$$
$$= \prod_{k=j,...,i-1}[p_k]*\sum_{k=j,...,i}[á_k], \quad \text{if } i=j+n-1 \text{ for all } 1<n\le N-j+1,$$
$$= \prod_{k=i+1,...,j}[m_k]*(\sum_{k=i-1,...,j}[p_k*m_{k+1}] + \sum_{k=i,...,j}[á_k * \sum_{t=k,...,j}[á_t]]), \quad \text{if } j=i+n-2 \text{ for all } 1<n<N-i+2,$$
$$= \prod_{k=j,...,i-1}[p_k]*(\sum_{k=j,...,i+1}[m_k*p_{k-1}] + \sum_{k=j,...,i}[á_k*\sum_{t=k,...,i}[á_t]]), \quad \text{if } i=j+n-2 \text{ for all } 1<n\le N-j+2.$$

(13) For k=1 there is $b_{N,t}(1)<b_{N,t+1}(1)$ for all t=1,...,N−1, $b_{N,t+1}(1)\neq 0$. The assertion $b_{N,t}(k)<b_{N,t+1}(k)$ for all t=1,...,N−1, $b_{N,t+1}(1)\neq 0$ and for a $k\ge 1$ may be true. Then there is for k+1:

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

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

for all t=1,...,N−1. Be $b_{N,t+1}(k+1)\neq 0$. Then there is:

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

because the m_t, $á_t$, and p_t are greater zero and $b_{N,t}(k)-b_{N,t-1}(k)\ge 0$, $b_{N,t+1}(k)-b_{N,t}(k)\ge 0$ and $b_{N,t+2}(k)-b_{N,t+1}(k)\ge 0$ under induction condition, the statement is true for all k.

(14) Derivation for S=256:
The **average of the not existing bytes δ** using 5000 iterations arises as follows:
(1) 99,86 =499302/5000 > δ (542. iteration),
(2) 94,34 =420672/(5000−541) > δ (705. iteration) without transient values (<542. iteration),
(3) 94,10 =404253/(5000−704) > δ (705. iteration) without transient values (<705. iteration)
⇒ δ=94 (705. iteration)
⇒ N=162 → 162/256 = 0,6328125 ≈ 63,28 %.
⇒ 705/162 ≈ 4,35 times average overlay of the N values for mZH being any value
⇒ >1410/x overlay events of a direct-mode dataset of x bytes length (N_0 „small“)
(4) mZH=3 (with the 1571. iteration)
⇒ 1571/162 ≈ 9,70 times average overlay of the N values for mZH=3
⇒ >3142/x overlay events of a direct-mode dataset of x bytes length (N_0 „small“)

(15) Derivation for S=65536:
The **average of the not existing byte-groups δ** using 300000 to 800000 iterations arises as follows:
(1) 24114,16 =12057106388/500001 > δ (359023. iteration),
(2) 24101,33 =10628156544/(800000−359022) > δ (360363. iteration) without transient values (<359023. iteration),
(3) 24101,28 =10595837949/(800000−360362) > δ (360363. iteration) without transient values (<360363. iteration)
⇒ δ=24101 (360363. iteration)
⇒ N=41435 → 41435/65536 = 0,6322479248 ≈ 63,22 %.
⇒ 360363/41435 ≈ 8,697 times average overlay of the N values for mZH being any value
⇒ >1441452/x overlay events of a direct-mode dataset of x bytes length (N_0 „small“)
(4) mZH=7 (with the 632086. iteration)
⇒ 632086/41435 ≈ 15,25488 times average overlay of the N values for mZH=7
⇒ >2528344/x overlay events of a direct-mode dataset of x bytes length (N_0 „small“)

(16) Derivation for S=16777216:
The **average of the not existing byte-groups δ** using 100000 to 250000 iterations at 1066 overlay events arises as follows:
(1) 6173447,99 =926023371853/150001 > δ (131212. iteration),
(2) 6172513,15 =733226664399/(250000−131211) > δ (133646. iteration) without transient values (<131212. iteration),
(3) 6172493,29 =718200456584/(250000−133645) > δ (133660. iteration) without transient values (<133646. iteration)
(4) 6172493,28 =718114040841/(250000−133659) > δ (133660. iteration) without transient values (<133660. iteration)
⇒ δ=6172493 (133660. iteration)
⇒ N=10604723 → 10604723/16777216 = 0,63209 ≈ 63,21 %.
⇒ 133660∗1066/10604723 ≈ 13,43567 times average overlay of the N values for mZH being any value
⇒ >854889360/x overlay events of a direct-mode dataset of x bytes length (N_0 „small“)
Example: 854889360 / 5935104 = 144 overlay events (5,66 megabyte dataset)
(5) mZH=9 (with the 220406. iteration)
⇒ 220406∗1066/10604723 ≈ 22,155486 times average overlay of the N values for mZH=9
⇒ >1409716776/x overlay events of a direct-mode dataset of x bytes length (N_0 „small“)
Example: 1409716776 / 5935104 = 237,5 overlay events (5,66 megabyte dataset)

Description of the Algorithm

Below there is: Extended 1.1.1 covers extended 1.1 covers extended 1, extended E is separate from these. The standard algorithm is called default. It is t≥1.

Program Initialisation:

1. The original key is repeated as multiple as 256**t t-byte-groups (i.e. t*256**t bytes) are reached or exceeded and cut forming the initial key.
2. Then one or more of these t-byte-groups of the initial key can be addressed and can be exchanged (**overlaid/overloaded**) by a given group-contents each.
   Every 2*t arbitrary characters/bytes next to one another, in which the first t characters/bytes are interpreted as an integer value with a value interval of 0 to 256**t –1 increased by 1 give the number of the referred t-byte-group of the initial key which should be overlaid by the following t characters/bytes giving the new content of the t-byte-group.
   In principle group positions may be overlaid multiple. After overlaying a position in fact, the overlay value is the new key value for the further overlay processing. The last overlay information applied to a position decides the value being the final key value at this position.
   Carrying out overlay processing, an input value or a dataset value first is transformed by
   overlay value := (input-/dataset-value + present key value ***** prime number
   + key value at previous position or at position 256**t for position 1) modulo 256**t
   with current **prime number**s :=113 | 30781 | 100000000019 for t=1|2|3,
   before the result-value is used as actual overlay value. Furthermore you can restrict overlay information taken into account to given position and value intervals, so you can prevent obvious overlaying using an “arbitrary” dataset. A single overlay information which is not in the given interval limits is ignored.

   In general you have to observe that for all positions in the position interval **in connection with an overlay information varying in any kind and being (nearly) infinitely long** the following is valid:
   As a result the key values of the key value interval are (almost) entirely transformed to overlay values of the overlay value interval.
   If there are overlay values not belonging to the key value interval, the following **more restricted rule** is valid:
   As a result all the key values are (almost) entirely transformed to overlay values of the overlay value interval which are not part of the key value interval.
   The better an overlay information complies with these rules, all the more the named conclusions are valid. That must be taken into account to avoid trivial keys.
   Given an initial key, an overlay information, and an overlay value interval, the **following is valid in general**:
   Any key value interval covering this overlay value interval provides the same result key, if no key value being element of the key value interval but not being element of the overlay value interval is found in the initial key.

   Choosing an arbitrary set of datasets ongoing enciphered, you can use them multiple to generate a key which doesn’t differ statistically from a “random key” (see "Convergence of Overlay Operations at Keys"). A possibly so changed initial key is used as the **real key**.

3. With the aid of this real key first modified in accordance with step 4 or subsequently with the aid of an operation key first modified in accordance with step 7 and step 10 above (in default form) a **predecessor** t-byte **S-table** and a corresponding t-byte **inverse S-table** for deciphering purpose is generated (see "Introduction/Summary", "Substitution Tables S").

4. The t-byte-group in a key (**pval**; see below) as from which 2* 2*7 bytes in the key are used to initialise both numbers of **each random number process** (for 52 bit mantissa under IEEE 754; for numbers having x mantissa bytes per number subsequently 7 bytes is exchanged by x bytes) is determined by the t-byte-value +1 at the
   - 1. (or following) position of the S-table using the current operation key as key,
   - 1. position of the real key using the real key as key.

   The operations being connected with the random numbers of a process act as follows:
   A. **Initialising** the random numbers of **process x**=1,2, only the least significant bits of the first key byte are used to fix the most significant bits of the mantissa of the 1. random number (IEEE 754 specific, different for other floating-point representations), analogously the least significant bits of the second key byte are used to fix the most significant bits of the mantissa of the 2. random number.
      a. Now the further key bytes (byte-group borders unconsidered) are alternately transferred to the 1. and 2. random number from most to least significant byte.
      b. The characteristics of both random numbers are modified so every characteristic represents the value zero.
      c. The result of the multiplication of the squared 1. with the squared 2. random number gets the new 1. random number while the new 2. random number is fixed to the squared original 1. random number. Then the characteristics of both random numbers are put back to zero.
      d. A **comparison value** (**compval[x]**) is calculated to locate the necessity of changing the current operation key (default) or changing of components (extended 1):
         compval[x] = ranvalpx and signbits
         signbits = bits to be compared, 2**(number of set bits in signbits) defines an average number of different values between identical compval-values
         ranvalpx = ranpxn1[b3]*256+ranpxn1[b4]
         | ((ranpxn1[b1]*256+ranpxn1[b2])*256+ranpxn1[b3])*256+ranpxn1[b4]
         for t=1|2 or 3,
         ranpx**n**y**[b**z**]** = byte z of y. random number of process x)
      e. Step 4.A.c is carried out again.
   B. **Applying** the random numbers to (real/operation) key or data, the following steps are repeated every 8 bytes (IEEE 754 specific) or for the remaining bytes of key and of data respectively:
      a. Both random numbers are squared.
      b. The least significant mantissa byte (by processing data the number of included mantissa bytes have to be a multiple of t) and the most significant mantissa half-byte of every random number are left out of consideration first.
      c. The following 4 (IEEE 754 specific) mantissa bytes each (from most to least significance) first of the 1. and then of the 2. random number are connected with key or data bytes being due by ‘xor’ operation if possible in consideration of the data length.
      d. The following (5.) mantissa byte of the 1. random number behind that is additionally connected by ‘xor’ operation with the 1. key or data byte already treated by the 2. random number if possible. In an analogues way the following mantissa byte of the 2. random number is connected with the 1. key or data byte already treated by the 1. random number.
      e. The 1. random number becomes the 2. random number, the 1. random number itself is replaced by the product of the old 1. and old 2. random number.
      f. The characteristics of both random numbers are put back to zero again.

g. If meanwhile the cycle of the current pseudo random number process should be passed already once, i.e. the mantissa bytes of the process are the same as at the beginning, then the following actions take place:
   i. The previous used **position value** (**pval**) of the t-byte-position in (operation or real) key as from which the last 2*7 bytes in the key are used to initialise both numbers of a new random number process is raised to (pval+7) modulo (t*256**(t–1) –7 +1).
   ii. A new pseudo random number process is initialised (step 4.A) and the so generated pseudo random numbers are connected further with the remaining key or data.

5. After raising position value pval to (pval+7) modulo (t/2*256**t –7 +1), the pseudo **random number process 1** and analogous **process 2** are initialised in accordance with step 4.A and the so produced pseudo random numbers are connected with the actual (operation) key by 'xor' operation in accordance with step 4.B giving an **modified operation key** (**mopkey**).
6. The modified operation key is transformed t-byte-wise by **S-table**.
7. In a loop of **m** runs (default: **m** =2; extended 1: **m** =2*t +mopkey[ranval1+1] mod (2*t+1) with
   [default] (1) **ranval1** =ranp2n1[b2] | ranp2n1[b2]*256+ranp2n2[b2]
   | (ranp2n1[b2]*256+ranp2n2[b2])*256+ranp1n1[b2] for t=1|2|3,
   (2) **ranval1** =ranp1n1[b2] | ranp1n1[b2]*256+ranp1n2[b2]
   | (ranp1n1[b3]*256+ranp1n1[b3])*256+ranp2n1[b3] for t=1|2|3,
   mopkey[x] = x. t-byte-group of modified operation key,
   the following operation key modifications take place:
   A. **'xor' from left to right**: mopkey[i+1] = mopkey[i+1] xor mopkey[i], i=1,...,256**t –1.
   B. t-byte-wise transformation via t-byte **S-table**.
   C. The **shift-value** in bytes is set to **svalb1** = mopkey[j]*t xor ranval2 with [(1) is default]
      (1) **j** = 256**t, **ranval2** = ranp1n1[b1] | ranp1n1[b1]*256+ranp1n1[b2]
      | (ranp1n1[b1]*256+ranp1n1[b2])*256+ranp1n1[b3] for t=1|2|3,
      (2) **j** = 1, **ranval2** = ranp2n1[b1] | ranp2n1[b1]*256+ranp2n2[b2]
      | (ranp2n1[b1]*256+ranp2n1[b2])*256+ranp2n1[b3] for t=1|2|3.
   D. **'xor' from right to left**: mopkey[i–1] = mopkey[i–1] xor mopkey[i], i=256**t,...,2.
   E. t-byte-wise transformation via **S-table**.
   F. The modified operation key is **moved cyclically** (the end of the operation key is regarded as adjacent to the beginning of the operation key) by **svalb1** bytes (see step 7.C).
8. The random numbers generated by the **random number process 2** are connected with the modified operation key by 'xor' operation in accordance with step 4.B.
9. The modified operation key is transformed t-byte-wise by **S-table**.
10. In connection with the current modified operation key **once again** the step 7 (with (2) instead of (1)) in form of the steps 7.D and 7.E followed by step 7.C (with (2) instead of (1)) also followed by the steps 7.A, 7.B and 7.F are carried out.

11. After generating an **predecessor S-table** in accordance with step 3 and setting **pval=1**, the 1. interim operation key is generated by running **steps 4 to 10 first time**.
12. Using this 1. interim operation key, the **real S-table** and **inverse S-table** are generated to use in the following (see "Introduction/Summary", "Substitution Tables S").
13. The **next run of the steps 4 to 10** generates a 2.interim operation key in which a t-byte-group is determined by the t-byte-value +1 at the 2. position of the S-table. From this t-byte-group the 2* 2*7 bytes are used to initialise the four numbers of **two random number processes** in accordance with step 4.A which are used to treat the data bytes.
14. **Another run of the steps 4 to 10** generates the **real operation key** as used for the data processing.

I. Initialisation of Data Processing:

15. For every separate treated dataset or data unit the real S-table and inverse S-table (see step 12), the pseudo random number processes for data processing (see step 13), the position value pval for initialising the random numbers (see step 4.B.g.i) and the real operation key (see step 14) are restored if no stream ciphering takes place.

II. Data Processing for each Data Block:

16. If the only data block of a dataset has less than t bytes, then running both an enciphering and a deciphering process these existing data bytes are connected separately with the mantissa bytes of the two random numbers of each pseudo random number process and with the bytes of the operation key using the 'xor' operation in accordance with the steps 20.C, 20.G, and 20.L. No further steps are carried out in this case.

17. Using variable **data block length** in bytes (**blklen**), this block length varies from the minimal to the double of minimal as maximal data block length. Using fixed data block length, minimal and maximal data block length are identical. The **minimal data block length** in bytes (**minblk**) has to be a multiple of **imb** = 4*t (=8 for t=1; see step 4.B too).
    A. If the last data block of a dataset is smaller than minblk and the conclusion of the two last data blocks is smaller than 2*minblk, then the conclusion of the last data block and the data block before last is treated as one data block.
    B. Else the data block length **blklen** is determined as between minblk and 2*minblk:
        blklen = (minblk/imb+(mopkey[ranval3+1] mod (minblk/imb+1)))*imb with
        ranval3 = ranp1n1[b1] xor ranp2n1[b1] | (ranp1n1[b1]*256)+ranp2n1[b1]
        | (ranp1n1[b1]*256+ranp1n2[b1])*256+ranp2n1[b1] for t=1|2|3
    C. If the last two data blocks reach or exceed 2*minblk and the length of the last data block is smaller than minblk, then the length of the last data block is incremented by a multiple of imb in this way that minblk is just reached or exceeded. In this case the length of the data block before last is reduced accordingly.

18. If the cycle of 1. current pseudo random number process should be passed already once during or at the end of processing the last data block (default) or if condition of step 19 is fulfilled (extended 1), then following actions take place at the beginning of the next/current data block:
    A. Extended 1.1: The current operation key is enciphered as data using a component exchange parameter dataset if existing.
    B. Running step 11 to 14 generates a new component record (see " Introduction/Summary ", "Change of Components") used for the data processing as of now.

19. If the bits in **currentval[1]** are equal to the bits in **compval[1]** (see step 4.A.d; default) with
    currentval[x] = ranvalpx and signbits
    (see step 4.A.d for random process x and with current random numbers),
    signbits = 2**q -1, q bits with 0<=q<$\log_2$(256**t/s) = 8*t -$\log_2$(s), in which s is the number of executions of the operation key per data block, i.e. s=blklen/(t*256**t)
    = 63 | 8388607 | 4294967295 at the moment for t=1|2|3,
    or if the current sum of blklen corresponding to current components is greater than t*256**t +1 (extended 1), then the following actions take place:
    A. Extended 1.1.1: The current operation key is enciphered as data using a component exchange parameter dataset if existing.

B. If **currentval[2]>currentval[1]**, then the steps 7.A to 7.F are processed with the current operation key in a loop of **m** runs (constant **m** =2 at the moment).
C. Else if **currentval[2]<=currentval[1]**, then the steps 7.D and 7.E followed by step 7.C (with (2) instead of (1)) also followed by the steps 7.A, 7.B and 7.F are carried out in a loop of **m** runs (constant **m** =2 at the moment).

20. If data should be **encipher**ed (in principle the terms "enciphering" and "deciphering" can be exchanged consistently in the entire text), then the following actions take place,
mainblk = blklen–(blklen mod t), **ranval4/5/6** generated at beginning of data block processing:
A. If blklen mod t ≠ 0, then the 2. and followed byte(s) of **each random number** (in general: t-1 bytes of the random numbers) of both relevant random processes are **stored** to use in step 20.E.c and step 20.O.c to process the last up to t-1 data block byte(s).
B. The (first) mainblk data bytes are transformed t-byte-wise via t-byte **S-table**.
C. Generating random numbers of the **1. random process**, the mantissa bytes of these random numbers are continually connected by 'xor' operation (step 4.B) with all mainblk data bytes.
D. If meanwhile the cycle of the current 1. pseudo random number process should be passed already once, then the following actions take place:
   a. The previous used **position value** (**pval**) of the t-byte-position in the operation key as from which the 2*7 bytes in the operation key are used to initialise both numbers of a new random number process is raised to (pval+7) modulo (t*256**(t–1) –7 +1).
   b. A new 1. pseudo random number process is initialised (step 4.A) and the so generated pseudo random numbers are connected by 'xor' operation with the remaining data of the mainblk data bytes further.
E. If blklen mod t ≠ 0, then with **block[x]** = x. byte of data block:
   a. The data block is enlarged by the byte before last till now and further bytes in front of it and so is completed to a **new t-byte-group** by movement and copy:
      block[blklen+t–(blklen mod t)–i] = block[blklen–i], i=0,...,t–1.
   b. This new t-byte-group is transformed via t-byte **S-table** (see 20.B).
   c. The bytes of the **1. random number process** stored in step 20.A are connected with the new t-byte-group by 'xor' operation (analogous to step 4.B).
   d. The bytes used for enlargement before are **copied back** to their initial positions:
      block[blklen–i] = block[blklen+t–(blklen mod t)–i], i= t–1,...,0.
   e. The bytes mainblk+1 to blklen are **connected** by 'xor' operation to **and replace** the bytes mainblk+1–t to blklen–t of the last (**n**.) t-byte-group:
      block[blklen–t–i] = block[blklen–t–i] xor block[blklen–i], i=0,...,(blklen mod t)–1.
F. The mainblk data bytes are transformed t-byte-wise via t-byte **S-table**.
G. The bytes of the **current operation key**
(a) by multiple processing of the operation key if blklen ≥ t*256**t,
(b) by processing of the 1. part of the next section (in length of the double of maximal data block length) of the operation key if blklen ≤ t*(256**t)/2,
(c) by processing the front part of operation key if t*(256**t)/2 < blklen < t*256**t
are continually connected by 'xor' operation with all mainblk data bytes.
H. Again the mainblk data bytes are transformed t-byte-wise via t-byte **S-table**.
I. In a loop of **m** runs (default: **m** =1; extended 1: **m** = mb+mf+mv with
   mb = 0 [basic part], **mf** = ($\log_2$(mainblk/t)+3) div 4 [fixed part],
   mv = mopkey[ranval4+1] mod (mf+1) [variable part]
   ranval4 = ranp1n1[b2] | ranp1n1[b2]*256+ranp1n2[b2]
   | (ranp1n1[b2]*256+ranp1n2[b2])*256+ranp2n1[b2] for t=1|2|3)
with **n** = blklen div t, the following modifications of the mainblk data bytes take place:

a. **'xor' from left to right**: block[i+1] = block[i+1] xor block[i], i=1,...,n–1.
b. t-byte-wise transformation via t-byte **S-table**.
c. **'xor' from right to left**: block[i–1] = block[i–1] xor block[i], i=n,...,2.
d. t-byte-wise transformation via t-byte **S-table**.
e. Extended 1: With the **shift-value** in bytes **svalb2** is set to
svalb2 = ((mopkey[j]*256**t)+mopkey[j+1]) mod blklen with
j = (ranval5+2*i) mod 256**t, j=0 <=> j=256**t for run i=1,...,m,
ranval5 = ranp1n1[b3] | ranp1n1[b3]*256+ranp1n1[b4]
| (ranp1n1[b4]*256+ranp1n1[b5])*256+ranp1n1[b6] for t=1|2|3,
the blklen data block bytes are **moved cyclically** (the end of the data block is regarded as adjacent to the beginning of the data block) by **svalb2** bytes.

J. Again step 20.G with multiple or the 2. part of the section or the back part of the operation key and step 20.H are processed (**current operation key**).
K. If blklen mod t ≠ 0, then:
a. The data block is enlarged by the byte before last till now and further bytes in front of it and so is completed to a **new t-byte-group** by movement and copy:
block[blklen+t–(blklen mod t)–i] = block[blklen–i], i=0,...,t–1.
b. This new t-byte-group is transformed via t-byte **S-table** (see 20.E.b).
c. The first element of the part of the **operation key** used in accordance with 20.G is connected by 'xor' operation.
d. Again the new t-byte-group is transformed via t-byte **S-table** (see 20.H).
e. The steps 20.K.b to 20.K.d are **processed again**.
f. The bytes used for enlargement before are **copied back** to their initial positions:
block[blklen–i] = block[blklen+t–(blklen mod t)–i], i= t–1,...,0.
L. Generating random numbers of the **2. random process**, the mantissa bytes of these random numbers are continually connected by 'xor' operation (step 4.B) with all mainblk data bytes.
M. If meanwhile the cycle of the current 2. pseudo random number process should be passed already once, then the following actions take place:
a. The previous used **position value** (**pval**) of the t-byte-position in the operation key as from which the 2*7 bytes in the operation key are used to initialise both numbers of a new random number process is raised to (pval+7) modulo (t*256**(t–1) –7 +1).
b. A new 2. pseudo random number process is initialised (step 4.A) and the so generated pseudo random numbers are connected by 'xor' operation with the remaining data of the mainblk data bytes further.
N. The mainblk data bytes are transformed t-byte-wise via t-byte **S-table**.
O. If blklen mod t ≠ 0, then:
a. The bytes mainblk+1 to blklen are **connected** by 'xor' operation to **and replace** the bytes mainblk+1–t to blklen–t of the last (**n**.) t-byte-group:
block[blklen–t–i] = block[blklen–t–i] xor block[blklen–i], i=0,...,(blklen mod t)–1.
b. The data block is enlarged by the byte before last till now and further bytes in front of it and so is completed to a **new t-byte-group** by movement and copy:
block[blklen+t–(blklen mod t)–i] = block[blklen–i], i=0,...,t–1.
c. The bytes of the **2. random number process** stored in step 20.A are connected with the new t-byte-group by 'xor' operation (analogous to step 4.B).
d. The new t-byte-group is transformed via t-byte **S-table** (see 20.E.b).
e. The bytes used for enlargement before are **copied back** to their initial positions:
block[blklen–i] = block[blklen+t–(blklen mod t)–i], i= t–1,...,0.

P. Extended E: With the **shift-value** in bytes **svalb3** is set to
svalb3 = ((mopkey[j]*256**t)+mopkey[j+1]) mod blklen with
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] for t=1|2|3,
the blklen data block bytes are **moved cyclically** (the end of the data block is regarded as adjacent to the beginning of the data block) by **svalb3** bytes.

21. If the data should be **decipher**ed, then the following actions take place:
A. Extended E: The cyclical move inside the data block in accordance with step 20.P is reversed.
B. If blklen mod t ≠ 0, then the 2. and followed byte(s) of **each random number** (in general: t-1 bytes of the random numbers) of both relevant random processes are **stored** to use in step 21.C.c and step 21.L.c to process the last up to t-1 data block byte(s).
C. If blklen mod t ≠ 0, then:
a. The data block is enlarged by the byte before last till now and further bytes in front of it and so is completed to a **new t-byte-group** by movement and copy:
block[blklen+t–(blklen mod t)–i] = block[blklen–i], i=0,...,t–1.
b. This new t-byte-group is retransformed via t-byte **inverse S-table** (see 21.D).
c. The bytes of the **2. random number process** stored in step 21.A are connected with the new t-byte-group by ‘xor’ operation (analogous to step 4.B).
d. The bytes used for enlargement before are **copied back** to their initial positions:
block[blklen–i] = block[blklen+t–(blklen mod t)–i], i= t–1,...,0.
e. The bytes mainblk+1 to blklen are **connected** by ‘xor’ operation to **and replace** the bytes mainblk+1–t to blklen–t of the last (**n**.) t-byte-group:
block[blklen–t–i] = block[blklen–t–i] xor block[blklen–i], i=0,...,(blklen mod t)–1.
D. The (first) mainblk data bytes are retransformed t-byte-wise via t-byte **inverse S-table**.
E. Generating random numbers of the **2. random process**, the mantissa bytes of these random numbers are continually connected by ‘xor’ operation (step 4.B) with all mainblk data bytes.
F. If meanwhile the cycle of the current 2. pseudo random number process should be passed already once, then the following actions take place:
a. The previous used **position value** (**pval**) of the t-byte-position in the operation key as from which the 2*7 bytes in the operation key are used to initialise both numbers of a new random number process is raised to (pval+7) modulo (t*256**(t–1) –7 +1).
b. A new 2. pseudo random number process is initialised (step 4.A) and the so generated pseudo random numbers are connected by ‘xor’ operation with the remaining data of the mainblk data bytes further.
G. If blklen mod t ≠ 0, then:
a. The data block is enlarged by the byte before last till now and further bytes in front of it and so is completed to a **new t-byte-group** by movement and copy:
block[blklen+t–(blklen mod t)–i] = block[blklen–i], i=0,...,t–1.
b. This new t-byte-group is retransformed via t-byte **inverse S-table** (see 21.D).
c. The first element of the part of the **operation key** used in accordance with 21.I is connected by ‘xor’ operation.
d. Again the new t-byte-group is retransformed via t-byte **inverse S-table** (see 21.D).
e. The steps 21.G.b to 21.G.d are **processed again**.
f. The bytes used for enlargement before are **copied back** to their initial positions:
block[blklen–i] = block[blklen+t–(blklen mod t)–i], i= t–1,...,0.
H. The mainblk data bytes are retransformed t-byte-wise via t-byte **inverse S-table**.

I. The bytes of the **current operation key**
(a) by multiple processing of the operation key if blklen ≥ t*256**t,
(b) by processing of the 2. part of the next section (in length of the double of maximal data block length) of the operation key if blklen ≤ t*(256**t)/2,
(c) by processing the back part of operation key if t*(256**t)/2 < blklen < t*256**t
are continually connected by 'xor' operation with all mainblk data bytes.

J. In a loop of **m** runs (default: **m** =1; extended 1: **m** = mb+mf+mv in accordance with step 20.I), the following modifications of the mainblk data bytes take place:
   a. Extended 1: The cyclical move inside the data block in accordance with step 20.I.e is reversed.
   b. t-byte-wise retransformation via t-byte **inverse S-table**.
   c. **'xor' from right to left reverse**: block[i–1] = block[i–1] xor block[i], i=2,...,n.
   d. t-byte-wise retransformation via t-byte **inverse S-table**.
   e. **'xor' from left to right reverse**: block[i+1] = block[i+1] xor block[i], i= n–1,...,1.

K. Again step 21.H, step 21.I with multiple or the 1. part of the section or the front part of the operation key and step 21.H once again are processed (**current operation key**).

L. If blklen mod t ≠ 0, then:
   a. The bytes mainblk+1 to blklen are **connected** by 'xor' operation to **and replace** the bytes mainblk+1–t to blklen–t of the last (**n**.) t-byte-group:
      block[blklen–t–i] = block[blklen–t–i] xor block[blklen–i], i=0,...,(blklen mod t)–1.
   b. The data block is enlarged by the byte before last till now and further bytes in front of it and so is completed to a **new t-byte-group** by movement and copy:
      block[blklen+t–(blklen mod t)–i] = block[blklen–i], i=0,...,t–1.
   c. The bytes of the **1. random number process** stored in step 21.B are connected with the new t-byte-group by 'xor' operation (analogous to step 4.B).
   d. The new t-byte-group is retransformed via t-byte **inverse S-table** (see 21.D).
   e. The bytes used for enlargement before are **copied back** to their initial positions:
      block[blklen–i] = block[blklen+t–(blklen mod t)–i], i= t–1,...,0.

M. Generating random numbers of the **1. random process**, the mantissa bytes of these random numbers are continually connected by 'xor' operation (step 4.B) with all mainblk data bytes.

N. If meanwhile the cycle of the current 1. pseudo random number process should be passed already once, then the following actions take place:
   a. The previous used **position value** (**pval**) of the t-byte-position in the operation key as from which the 2*7 bytes in the operation key are used to initialise both numbers of a new random number process is raised to (pval+7) modulo (t*256**(t–1) –7 +1).
   b. A new 1. pseudo random number process is initialised (step 4.A) and the so generated pseudo random numbers are connected by 'xor' operation with the remaining data of the mainblk data bytes further.

O. The mainblk data bytes are retransformed t-byte-wise via t-byte **inverse S-table**.

Examination of the Safety of the Algorithm in Brief

III. Single Components and Steps

The following simple or combined step numbers refer to the step information in the document "Description of the Algorithm" above and first the default case is discussed.

The ciphering method used here consists of 3 components which are used in combination and are dependent on each other:

1. The first component is the key and the operation key derived from the key respectively. This component is the starting component of the construction process of the other two components too.
2. The S-table and the inverse S-table deciphering data respectively constitutes the second component. This component is (both tables respectively are) generated by the key and the current operation key respectively.
3. The third component is formed by two continual random number processes as well initiated by the key and the operation key respectively.

4. To raise the degree of the effectiveness of the S-tables in particular using short keys, the key string is expanded to the full length of 256**t t-byte-groups, i.e. the original key string is duplicated multiple and the result string is truncated to 256**t t-byte-groups.

5. Because there could be some problems to enter a key of just only about t*256**t bytes and the repetition of the original key isn't a real solution to this problem, a method described in "Introduction/Summary", "Keys(1)" and more differently in step 2 is integrated which removes the uniformity of the key generation process. Further it opens the opportunity to use the contents of any dataset as an additional source of key generation.

6. The random numbers in " Introduction/Summary", "Pseudo Random Processes" and in step 4 are responsible for the "noise" because in general the data has not to be dissimilar. Only the dissimilarity make it possible that the S-table works in wide range of sections and therefore has the maximum effect. Also a requirement of the special effect of the 'xor' operation is that the data isn't to similar.
   Apart from this the random numbers are responsible for the (with regard to the powerfulness of variety not so significant) "basic security" of the data too that is why random numbers here of the 2. pseudo random number process are used again at the end of the ciphering process ("Introduction/Summary", "Definition of the Cipher Envelope" and step 20.L) besides the S-table.
   Further the random number bytes trigger the variability and fortuity of the ciphering process. This means variable block length, number of cycle runs, shift values, and comparison values are generated by random number bytes as seen in steps 4.A.d, 7, 17.B, 19, 20.I and 20.P for example.
   Using random numbers in accordance with step 4.B, importance is attached to the fact that possible side effects aren't detrimental to the "stochastic". That is why only core bytes of each (pseudo) random number are used and step 4.B.d guarantees the coupling of the both random numbers of a random number process and the integration of the remaining core bytes too.

Further importance is attached to the fact that a random number process does not repeat. Every time this should take place, it is changed over to a new random number process to prevent targets triggered by identical operators and possibly similar or identical blocks.

If this event occurs in connection with the 1. random number process treating data, all components are new generated ("Introduction/Summary", "Change of Components(2)" and step 18). So in principle it takes place a complete key change form the present key to the current operation key.

7. The action which is described in step 7 is used both in connection with the operation key and in modified form (see steps 20.I and 21.J) in connection with data. An explanation of these actions are given in "Introduction/Summary", "Kernel Explanation (2)"

   The shift operation in step 7.F makes a direct statistical attack concerning the operation key more difficult.

   Also the operation key connection with data (see steps 20.G and 20.J) guarantees the real influence of each bit of the (current operation) key to each bit of a data block.

8. S-tables, random numbers, as well as the “shredder” process described before are used not only with data blocks but also with modifications of operation keys.

   If generating new S-tables and inverse S-tables (steps 12 and 18.B) and generating new pseudo random number processes (steps 13 and 18.B), beside the new operation key (steps 14 and 18.B) separate interim operation keys are generated and used (step 11, 13, and 18.B) to make conclusions and therefore attacks regarding pseudo random numbers, S-table(s), and operation key in connection with the data treatment of these components much more difficult.

   The operation key (in connection with data) is permanently applied to the data (step 20.G, 20.J, 21.I, and 21.K) during the ciphering process.

   The operation key is changed before reaching the nth application (step 19) to prevent that it could be possible to conclude something about the total operation key by multiple application. Here n is the number of byte-groups in the operation key =256**t. On the other hand just for expenditure reasons it is taken pains to make only necessary modifications and to avoid new targets of deducing the operation key.

   A coupling of the operation key development is produced to the development of the 1. random number process by reference to this process (step 19) regarding not only the “when” but also the “how”.

9. In general the (inverse) S-table is used among each two adjoining ‘xor’ operations to bring in the mightiness of the possible S-tables recently at any changed situation and to “protect” the operation phases to each other.

IV. Worst-Case Scenarios

The following steps are carried out schematically **generating the components**:

A. 1. interim operation key
B. S-table and inverse S-table
C. 2. interim operation key
D. pseudo random number processes (initialisation; for data)
E. real operation key

in which **operation keys** are generated as follows (without special features):

(1) 'xor' of the random numbers of the 1. random number process (key)
(2) S-table (B for C, D, and E, predecessor of B for A)
(3) 'xor' of key data from left to right
(4) S-table (B for C, D, and E, predecessor of B for A)
(5) 'xor' of key data from right to left
(6) S-table (B for C, D, and E, predecessor of B for A)
(7) cyclical shift (in accordance with the result of (4) and relevant random number)
(8) repetition of the steps (3) to (7)
(9) 'xor' of the random numbers of the 2. random number process (key)
(10) S-table (B for C, D, and E, predecessor of B for A)
(11) repetition of the steps (3) to (8), step (3) and (5) transposed.

By **ciphering a block** following schematic steps are executed which are processed in inverse order with inverse S-table by deciphering:

a. S-table
b. ‚xor' of the random numbers (1. random number process; data)
c. S-table
d. ‚xor' of the operation key (or of the 1./front part of the operation key)
e. S-table
f. ‚xor' of data from left to right
g. S-table
h. ‚xor' of data from right to left
i. S-table
j. ‚xor' of the operation key (or of the 2./back part of the operation key)
k. S-table
l. ‚xor' of the random numbers (2. random number process; data)
m. S-table

So be T the (byte-group-)S-table,
ZZ_1 the random number elements of the 1. random number process in block length,
ZZ_2 the random number elements of the 2. random number process in block length,
Op_A the/the 1. part of the/the front part of the operation key,
Op_E the/the 2. part of the/the back part of the operation key,
E_o the upper triangular (byte-group-)matrix with value 1 at all non-zero positions,
E_u the lower triangular (byte-group-)matrix with value 1 at all non-zero positions,
$\oplus$ the byte-group-wise addition without carry over (‚xor'),
$\otimes$ the (vector) multiplication which belongs to $\oplus$,
then ciphering a block can also be described as follows:

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

The scenarios described in the following are fictitious for the moment i.e. first of all it is of no importance whether they can arise in reality or not. It is just intention to demonstrate the high safety of the ciphering system even if – however possible – the components are assumed to be "well-known".

Naturally the original key is assumed to be unknown in all scenarios.

10. Let us look at the scenario knowing the random number processes for data as well as the S-table as well as the original data and the enciphered data too. The **real operation key** is assumed to be **unknown**.
    1. Then you can reach step d starting off on the original data, starting off on the enciphered data the steps m, l, k, and j are reachable.
    2. Most parts of the 1. interim operation key can be derived from the S-table in fact but a considerable part remains unknown.
    3. Already the 2. interim operation key isn't derivable in direct manner because, operating step (5), the unknown information of the 1. interim operation key affects the known parts of it in a severe way. Step (7) in connection with the 2. interim operation key is unknown too because here the highest byte-group of the 1. interim operation key which even couldn't derived has influence. For this however all possible variants could possibly be tested. But the 2. interim operation key at that moment isn't comprehensible any longer at the latest if the steps (3) to (7) (step (8) and twice in step (11) again) are repeated.
    4. It's true that the component D is known but only less elements of the 2. interim operation key are derivable about this. If the steps (3) and (5) are repeated on the 2. interim operation key at the latest, the real operation key isn't derivable in direct or indirect manner any more.
    5. With that it can not be comprehended how to reach step d starting off on step j or vice versa without inspecting all stats of the real operation key. Analytical/statistical examination trials using several or all data blocks are made fail because the operation key is modified by the steps (3) to (7) and by the steps (5), (6), (3), (4), (7) respectively just before the possibility hypothetically would be given to deduce this operation key. Even the knowledge of "when" and "how" by knowing the random number processes doesn't help in the process because the steps (3) to (5) combined with (4) or (6) destroy any form of partial knowledge.
    6. Result: The real operation key isn't deducible using analytical or statistical methods (as it is obvious up to now).

11. Let us look at the scenario knowing the random number processes for data as well as the real operation key as well as the original data and the enciphered data too. The **S-table** is assumed to be **unknown**.
    1. Original and enciphered data are not modifiable in a direct way without S-table.
    2. Viewing the above steps from the real operation key to the 2. interim operation key in reverse direction, it is immediately obvious that the possibility to pursue the real operation key back to the 2. interim operation key ends at step (6) at the latest.
    3. Hypothesis of values of single elements in the S-table aren't helpful because, using steps (5) and (3), mutual connections are taken place with the remaining results of the S-table which themselves are starting positions using the next and previous operation respectively.

4. Even if only a very small part of the S-table is used really, these assumptions aren't helpful because these parts remain unknown.
5. The well-known pseudo random number process only makes it possible to uncover less positions of the 2. interim operation key. To conclude the 1. interim operation key from that, it hits the same obstacles as the trail to infer the 2. interim operation key from the real operation key.
6. Result: The S-table isn't deducible using analytical or statistical methods (as it is obvious up to now).

12. Let us look at the scenario knowing the S-table as well as the real operation key as well as the original data and the enciphered data too. The both **pseudo random number processes** for data are assumed to be **unknown**.
    1. Viewing the ciphering of the data blocks, it is found out that the trial of collecting information about the random number processes ends first at step b and l in spite of the knowledge of S-table and inverse S-table and of the operation key.
       Single positions inside the data blocks are not separable because of step f and h (the 'xor' operation has the effect of a remainder class formation in this sphere) i.e. both random number processes should have carried out entirely in a enumerative way to determine these processes accordingly.
    2. If you would derive the pseudo random number processes (component D) from the operation keys, the 2. interim operation key has to be well-known at least at that positions which are relevant to the processes.
    3. Most parts of the 1. interim operation key can be derived from the S-table in fact but a considerable part remains unknown.
    4. The 2. interim operation key isn't derivable in direct manner from that because, operating step (5), the unknown information of the 1. interim operation key affects the known parts of it in a severe way. Step (7) in connection with the 2. interim operation key is unknown too because here the highest byte-group of the 1. interim operation key which even couldn't derived has influence. For this however all possible variants could possibly be tested. But the 2. interim operation key at that moment, also in parts, isn't comprehensible any longer at the latest if the steps (3) to (7) (step (8) and twice in step (11) again) are repeated.
    5. The trial to derive the random number processes from the real operation key immediately hits the problem in step (9) to get the random number process for the transition from the 2. interim operation key to the real operation key. As described in step 13, the start position of the 14 bytes generating the shown random number process in the 2. interim operation key is known in fact but these bytes themselves are part of the 2. interim operation key being unknown. Without this random number process there is no possibility to comprehend neither the steps (1) and (9) nor the steps (7) and therefore (8) to say nothing to the "reversal" of the steps (3) and (5) and their repetitions in step (8).
    6. Result: The pseudo random number processes for data aren't deducible using analytical or statistical methods (as it is obvious up to now).

13. Further scenarios as the knowledge of the 1. interim operation key etc. are conceivable but doesn't really make sense. The scenarios shown here are used to prove with known or possibly known elements as enciphered data and original data on the one hand and with knowledge of S-table, pseudo random number processes, and real operation key which is possibly attained by data analyses using statistical methods too on the other hand that a high safety can be still guaranteed.

14. The result of this examination in brief of the default case is that you have to had either the original key or really all of the three components (B, D, E) mentioned before to comprehend the whole ciphering process. **Two of these named components aren't sufficient to deduce the missing component.**

15. Even if all of the components B, D and E and therefore the entire ciphering process should be known, it's clear looking at the unknown S-table, the unknown pseudo random number processes and the "shredder" process, all deduced by the original key, that you can hardly extract the original key from these deduced components in a direct way.
    Even the original key is effectively "protected" against its own ciphering process.

16. It can be seen as a consequence of this examination too that the key space can not be seen as to be reduced to the S-table. In fact all 256**t positions of the key i.e. t*256**t bytes are effective. Therefore the key space has (256**t)**(256**t) elements which can be addressed by the shown algorithm minimum in most parts.
    Whether the total parameter space shown in "Introduction/Summary", "Size of Parameter Space" is usable by transition to the next component record changing only some but not all components, have to be reserved to future explorations.

V. Necessity of the given Components and Steps

The following reflections can give reasons for the necessity of the given components and steps i.e. these components and steps are not spare:

(a) The usage of the steps f. to i. ciphering a block isn't sufficient to avoid identical destination blocks using identical original blocks. It's rather necessary to integrate an ongoing "random" process generating the destination blocks. For example this can be done by using an ongoing operation key or by integrating a pseudo random number process.

(b) If only an operation key is used, its elements have to be connected with **all** block elements at the beginning of ciphering a block. This have to be done in such manner that there aren't many block elements connected with one operation key element and there isn't any block element not connected with an operation key element. Disregarding these conditions it's the possibility to get 4-times alternated or alternated or identical values in the destination block using identical input data elements in connection with the steps f. to i.

(c) Using the operation key as whole or parts only at the end, it is **not** sufficient because this results in value **T**(0) for every 4. block element after using step f. to i. and for every S-table T. So these parts of the operation key elements y can be evaluated as "**T**(**T**(0) **xor** y)" and therefore are "well known".

(d) It isn't sufficient too using an operation key at the beginning which doesn't totally cover up a data block and additionally using a pseudo random number process at the end. Here also it is possible to calculate 1/4 of the pseudo random number bytes and you can possibly deduce the whole random number process.

(e) In general, using the operation key as the only "random" process, the length of the operation key has to be greater or equal than the length of any data block. Furthermore the operation key has to be modified again if it is work worn. To do so you have to pay attention not to modify the operation key in such way that, for example by using data blocks with binary zeroes, identical destination blocks arise or parts of destination blocks are identical to any operation key parts themselves.

(f) If you only use a pseudo random number process, you have to insert this process at the beginning of the block modification as described before. Also this process has to modify **all** data block elements.

(g) If you only integrate a "random" process at the beginning there is a risk that this "random" process can be evaluated if the destination data is very homogeneous by chance (even though with an extreme small probability). To confirm this proposition you can apply the previous argumentation in reverse order and by using the reverse S-table. To avoid this (rare) situation you have to establish a "random" process at the beginning and at the end of the block ciphering algorithm both totally cover the entire data block.

(h) Not even generating a “random” process the generation of the operation key is a necessity because any renewal of the ciphering process starts with this operation key if for instance the (main) pseudo random number process is imminent to be periodic. In addition the safety against compromising of former ciphered data, even if all current system components are compromised, is connected with the renewal of the system and therefore with the operation key (see IV., a) below). Furthermore the operation key represents in a way the powerfulness of the used S-table and therefore represents a more essential safety than 14 bytes of a random number process.

(i) It remains the periodicity of the operation key changed like this as an open point however. Examinations of the author turn out that only n of n! possible S-tables with n=2**m elements have the characteristic to be totally associative in accordance with the ‘xor’ operation i.e. **T**(y **xor** z) = **T**(y) **xor** z for all y,z.

Here additional examinations have to be certainly done (if not already carried out) to get a more exact knowledge of the periodicity of the so generated operation keys.

Because the periodicity of an operation key permanent modified by using a S-table stays unsolved for the moment, the current algorithm uses both “random” processes – in modified form looking at the operation key (see step 19) – in the steps a. to e. and j. to m. to guarantee a “very important” cycle length in combination with a high safety against compromising.

VI. Extensions to increase the Safety

To increase the safety of the default algorithm especially in connection with professional applications, further modifications of the standard algorithm present themselves and their advantages are discussed more exactly in the following:

1) To look at the default algorithm the exchange of the components B, D, and E takes place just when the cycle of the 1. current pseudo random number process should be passed already once. Instead of the simple modification of the current operation key a complete exchange of these components is made (extended 1). Therefore a special variable (here: 'SignBits') determines the average amount of data blocks to be processed before a new complete component exchange takes place (step 18/19).

2) Every block is used in connection with its own operation key or operation key section (extended 1). If exhausted, the current operation key is modified in accordance with the steps (3) to (8) or the step (11) each (step 19).

3) The steps (8) and (11) of the operation key modifications are repeated m times (extended 1, step 7), in which m is determined by the current pseudo random number process and the current operation key respectively limited by upper and lower bounds.

4) The steps f to i of the data block modifications expanded by a step (extended 1, step 20.I.e) analogous step (7) are repeated m times (extended 1, step 20.I), in which m is determined by the current pseudo random number process and the current operation key respectively also limited by upper (2***mf**) and lower (fixed part **mf**) bounds.

5) A separate generation (extended 1.1) /modification (extended 1.1.1) key is used to change the operation key. If it is a component exchange in accordance with 1) or a operation key modification in accordance with 2) respectively, the current components in accordance with B, D, and E of this separate key are used to get a new operation key by enciphering the current operation key each time.

6) A cyclical byte-shift of each data block takes place after enciphering (extended E, step 20.P).

The extensions mentioned before are motivated by the following considerations:

a) Because every exchange of the components is carried out as an initial initialisation using the current operation key instead of the original key, there is no “simple” possibility to conclude the preceding components by examination of the current components.
   If a set of components is compromised, so the data blocks of **former** stets of components are **not** compromised.
   The “safety” of the algorithm is determined by the special variable (’SignBits’) described in modification 1). In extreme case it’s possible to exchange the components after any ciphered data block.
   To reduce the time needed, it’s possible to modify the operation key in accordance with 2) in a “simple” manner between two component exchanges. The disadvantage is that if such a set of components is compromised all data blocks of following sets of components are compromised too.

b) The linear operations in accordance with modification 3) and 4) are repeated m times to expand the effectiveness of the S-tables to bigger regions of these tables (especially by using “small” block lengths in comparison with the table length) and to intensify the mix-up of data during ciphering blocks and generating components and modifying operation keys respectively.
   If the involved cycle lengths of the current S-table aren’t smaller than the repetition factor m for the most part, then a block first comprising exactly one t-group element (repeated correspondingly) is transformed back after $n=\log_2$(amount of t-byte-group elements in block)/2 runs into a block which average amount of different elements corresponds with that of a chance block of same block length (doubling twice of the different elements per run using a “suitable” S-table).

c) To prevent compromising the next set of components and therefore all data ciphered by this component set, if the current set of components is compromised, **using stream ciphering** there is the possibility to insert a separate generation and modification key respectively to modify the operation key in accordance with modification 5). With the aid of this separate key an ongoing chain of generation components in accordance with A to E is generated and supplied by a component exchange parameter dataset only to encipher the operation key as a pseudo data block at the beginning of an exchange of components and of the operation key respectively. Then the “enciphered” operation key is used to carry out the exchange of the components in accordance with the steps A to E and the steps (3) to (8) or step (11) respectively.
   If a set of components is compromised using this method just described, then the data blocks of **subsequent** sets of components are **not** compromised.
   To compromise this “series” of generation key components, “very much” normal sets of components have to be compromised first or the set of components of the generation key is “well-known” past a certain time (the generation key itself is unknown, if you are working only with components; by the way there is no necessity that these initial components have to be generated by only **one** key).
   With that all data blocks are well-known past this time, but in this case data blocks which are handled with former components are **not compromised too.**

d) If the messaging service is organised so that there is a headquarter/server on one hand and there are branches/clients on the other, every client can be assigned its own generation key i.e. the different message routes can be protected in an independent and individual way.
Because the ciphering method progress is **not** coupled to block contents and is **not** depended on enciphering or deciphering data using the current set of parameters, there is the possibility to use (straddling) stream ciphering in two directions simultaneously (but not parallel for **one** set of parameters) fixing the sequence. This development of the components and parameters goes in absolute parallel manner on the client side as well as on the server side with complete safety and **without any additional key exchange.**
Since the parameter dataset is internally designed in this way that
- all conditions of a parameter dataset are valid regular conditions,
- you cannot distinguish enciphered from deciphered parameter datasets, and therefore
- any "wrong" deciphered parameter dataset is an able to work parameter dataset,

the client can "deciphere" a parameter dataset with a **"compromising" key** instead of the regular key indicating "client is compromised" to the server **without any technical possibility for a third person to find out.** With that the server can change to the "compromising" parameter dataset itself past this time and it can transfer "irrelevant" information to the client to deceive third persons or to put third persons off the scent without possibility of direct detection.

e) All **message exchange** mentioned before can use **Internet**. Because a faulty data block **doesn't** affect other data blocks and their enciphering and deciphering respectively, a repetition of contents and/or of blocks is possible at any time. If the described safety to be connected with compromising components and data should be retained entirely, it's better to give the repetition of contents priority.

f) It can be useful to establish a cyclical byte-shift in accordance with modification 6) to make statistical attacks more difficult or impossible using fixed block length.

Extension and Application of the Algorithm

Example of the Application of the Algorithm in the Sphere of Agents

In the following it is presupposed that the entire communication is carried out by stream ciphering using CODING with “normal” and component exchange parameter datasets and every agent has its own set of parameter datasets however she/he is got it. Next it is presupposed that every communication between the agents takes place using one (or more) headquarter(s). The direct communication between the agents isn’t useful because it’s dangerous in accordance with the compromising of agents to say nothing of technical problems in connection with that.

First there has to be an additional component exchange at the end of an offline process.

If an agent is busted among two offline processing tasks, first there are no consequences in accordance with the safety . The parameter datasets represent the state straight **after** a component exchange using stream ciphering, therefore if the agent is forced to activate the parameter datasets he simply uses his compromising key and the headquarter is informed. The former blocks couldn’t be deciphered or reproduced even using the correct (“normal”) keys of the agent (!) as described in the “Examination of the Safety of the Algorithm in Brief”.

To get safety-relevant information, the enemy has to copy the parameter datasets and then he has to wait for the next block sent (i.e. the first block in connection with the copied parameter datasets) and copies it too. Then the enemy has the possibility to crack the keys of the parameter datasets by intelligent “brut force”-attacks where the first block is only used to check to have found the right keys.

To look at the algorithm realized up to now, the weak point of “simple” keys protecting parameter datasets disturbs. Because such a key has to be easy to remember intentionally, overlay datasets should be used additionally to secure parameter datasets. This gives much more safety against intelligent “brut force”-attacks, if the enemy acquires these parameter datasets.

Even if the enemy has the possibility to get the parameter datasets, the problem remains that the agent must not realize this. If the agent notices anything, he doesn’t send any regular block but only sends compromised blocks. So the enemy couldn’t get any relevant information from the headquarter. In addition the enemy has the problem to copy all parameter datasets at the same time. If he doesn’t copy one parameter dataset before using a set of parameter datasets, all copied parameter datasets of this used set of parameter datasets are worthless (!).

This means for the handling that the used parameter datasets should be located on different media and have to be brought together only for ciphering. For example the hard disk of the ciphering computer, a memory stick which is carried with the agent and a third rewritable medium stored in a safe or e.g. hidden in the apartment of the agent is suitable. Let us additionally assume that the agent usually reports every day to the headquarter, then the enemy has about 24 hours to copy all information stored at these three media without making the agent notice anything (!). If the agent has only any idea that there is something wrong, he will not generate a regular ciphered block never again and the enemy has no possibility to note that, so the parameter datasets are worthless (!).

Referring to the existing algorithm it means that a third parameter dataset has to be integrated if necessary to facilitate the third medium described before.

Conversely it is necessarily the case that the agent has to detect his compromising in least doubt. If the enemy takes possession of all parameter datasets and the corresponding first block, he has the possibility to force the agent to give the correct keys instead of "brut force"-attacks because the enemy can check the keys given by the agent with this first block every time (!).

Here it turns out that the offline process could be shown to be inevitable. Using an online process by the agent there is the possibility that for example the enemy escapes the electricity after the agent has sent the first block and so the enemy has all needed parameter datasets possibly in unchanged format if there isn't a component exchange after every block immediately stored to the parameter datasets. Naturally this is avoided by using batteries, but the enemy can simply assault the agent exact after the dispatch of the first block.

In offline mode it's much more difficult to realize such an attack. Outwardly there is no clue for this moment (if the screen contents of the ciphering computer isn't just readable from outside; if so, the enemy could directly read the information without any key !) and the information to be sent could be generated at one time and could be sent at later time. At send-time it's too late for the enemy because the parameter datasets are ciphered again after generating the information. Using additional sophistications not discussed here you can safely design these problems even in online mode (!) so the enemy has no possibility of intervention (unless the enemy has immediately the possibility to stop the computer and to copy the main storage).

To decipher the information from the headquarter the agent has to activate the parameter datasets again and therefore has to use the regular keys. This facilitates the enemy to attack the agent now at this moment. But this situation is "unproblematic" for the agent, if there is an independent second set of parameter datasets only used to receive the information from the headquarter. The regular keys of the parameter datasets can be handed over by the agent, if he can be sure of sending the next information by parameter datasets processed by the compromising keys to the headquarter. If this is possible, then the headquarter can identify the compromising state of the agent and further will send only "trivial" information to the agent. Former received blocks are never compromised (!; see the examination of the safety).

It cannot be avoided that the last information from the headquarter if received from the agent is well-known to the enemy because the enemy can force the deciphering of this message under full control of the enemy. It's possible the agent can manipulate the information process but under the high attention and the technical possibilities of the enemy it doesn't seem to be possible but very doubtful at least.

If the agent detects that he can't decipher received information from the headquarter any more he sends his parameter datasets of incoming information to the headquarter. Because the enciphered parameter datasets will be enciphered again and these parameter datasets "only" concerns received information traffic of the agent, there is no problem to do this. The headquarter will analyse the arriving parameter datasets and will change its parameter datasets to the status of the datasets of the agent. If the fault remains, the agent can inform the headquarter about these facts and can accept the disturbed incoming communication. That means, he cannot get additional information from the headquarter but he can still send and can try to change this situation immediately or later again.

The enemy however can not make any more use of this conduct than described before, because the sent parameter datasets “only” concerns incoming information from the headquarter.

If in reverse proportion the headquarter isn’t able to resynchronise with the send-site parameter datasets of the agent in spite of efforts, it will send the agent its receive-site parameter datasets with the next reply for the agent to use for outgoing messages. This also is “unproblematic”, because these parameter datasets are enciphered parameter datasets which “normally” could be found by the agent in the same state too. If the fault remains, the headquarter will send an corresponding enciphered message to the agent but it will avoid “secret” message contents in future because it cannot exclude the compromising state of the agent. Then the Agent has the possibility to check his outgoing keys (possibly including overlay datasets). If he isn’t able to change the situation (in spite of repetitions), he will discontinue the communication with the headquarter and will see himself as “compromised”.

It is emphasized once again we assume that the enemy has detailed information about

- **all discussed (also internal) arrangements and rules both in accordance with the agents and in accordance with the headquarter**,
- **the algorithm**, and
- **the entire communication traffic between agents and headquarter(s)** and is able to tap and record this traffic completely.

Nevertheless using the rules of conduct described before, the enemy only selectively has the possibility to get less secret information, if the agents exercise attention correspondingly, and this without any change of keys.

It’s not the interest of this examination to discuss the possibilities of the agents to hide their locations and further more topics. Because it’s possible to connect to the internet via handy etc., there are no fundamental restrictions and therefore no problems in this connection for the moment.

Using possible situations, it just should be discussed which chances in fact are opened up by CODING in practice. Naturally it’s also true to protect online connections (e.g. via “Voice over IP”) with the program CODING or to use handy hardware to operate. The exact arrangements have to be discussed in more detail for this however.

- - -