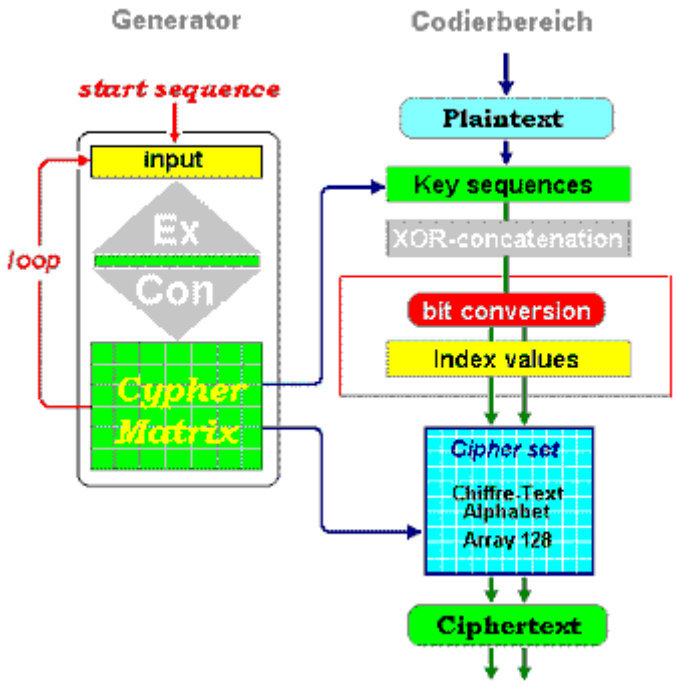
„Codegraphy“ as Part of Cryptography

(Ernst Erich Schnoor)

With **CypherMatrix** procedure – named by the autor – new connections will be disclosed in cryptography. Caused by „bitsystems“ and „bitconversion“ new levels and functions arise which will take the place of previous „primitives“ [#1]. The current cryptography is pure letter codification [#2] within the scope of bitsystem on base 8. The CypherMatrix procedure overlaps this coverage and realizes all bitsystems from base 1 up to base 16.

Encoding is very easy:

> A **generator** produces necessary system parameters
> and cipher text is written in the **coding area.**



Both areas are combined together, but can be used separate and stand-alone, as well.
In order to explain the connections we have to trace back to the main features of digitally techniques.

1 Systematizing Bit Series

In order to work with bit series systematically they have to be systemized, i.e. scaled and devided into definite segments (units). Up to now bit series are not scaled, Comparable to numerical theory bit series can be systematized in a significance order system, too.

system alphabet

bit sequences:	1-bit	= bit system on base 1	=	2^1 signs	= 2 units
	2-bit	= bit system on base 2	=	2^2 signs	= 4 units
	3-bit	= bit system on base 3	=	2^3 signs	= 8 units
	4-bit	= bit system on base 4	=	2^4 signs	= 16 units
	5-bit	= bit system on base 5	=	2^5 signs	= 32 units
	6-bit	= bit system on base 6	=	2^6 signs	= 64 units
	7-bit	= bit system on base 7	=	2^7 signs	= 128 units
	8-bit	**= bit system on base 8**	**=**	**2^8 bytes**	**= 256 bytes**
	9-bit	= bit system on base 9	=	2^9 signs	= 512 units
	10-bit	= bit system on base 10	=	2^10 signs	= 1024 units
	11-bit	= bit system on base 11	=	2^11 signs	= 2048 units
	12-bit	= bit system on base 12	=	2^12 signs	= 4096 units
	13-bit	= bit system on base 13	=	2^13 signs	= 8192 units
	14-bit	= bit system on base 14	=	2^14 signs	= 16384 units
	15-bit	= bit system on base 15	=	2^15 signs	= 32768 units
	16-bit	= bit system on base 16	=	2^16 signs	= 65536 units
	32-bit	= bit system on base 32	=	2^32 signs	= 4294967296 units

A bit system consist of a definite number of signs which are aggregated in an appropriated system-alphabet. The fundamentally connections are shown in following scheme:

System base	System-alphabet	Bit series in Units indexing	Lengths ratio
bitsystem on base 1	2	0 1 0 0 1 1 1 0 0 1 1 0 1 1 1 1 0 1 1 1 0 1 0 0 0 1 1 0 0 0 0 0.	1:8
bitsystem on base 2	4	01 00 11 10 01 10 11 11 01 11 01 00 01 10 00 01 01 10 00. 1 0 3 2 1 2 3 3 1 3 1 0 1 2 0 1 1 2 0	1:4
bitsystem on base 3	8	010 011 100 110 111 101 110 100 011 000 010 110 001 ... 2 3 4 6 7 5 6 4 3 0 2 6 1	1:2,66
bitsystem on base 4	16	0100 1110 0110 1111 0111 0100 0110 0001 0110 0010 ... 4 14 6 15 7 4 6 1 6 2	1:2
bitsystem on base 5	32	01001 11001 10111 10111 01000 11000 01011 00010 01... 9 25 23 23 8 24 11 2	1:1,6
bitsystem on base 6	64	010011 100110 111101 110100 011000 010110 001001 ... 19 38 61 52 24 22 9	1:1,33
bitsystem on bases 7	128	0100111 0011011 1101110 1000110 0001011 0001001 ... 39 27 110 70 11 9	1:1,143
bitsystem on base 8 *operations*	256	01001110 01101111 01110100 01100001 01100010 011.. 78 111 116 97 98 4E 6F 74 61 62 N o t a b ... *streamcipher, blockcipher with congruence of length, Feistel-nets* ECB. CBC. CFB. OFB. DES. IDEA. AES, RSA, *differential and lineare cryptanalysis, and other programs*	1:1
bitsystem on base 9	512	010011100 110111101 110100011 000010110 001001100 156 445 419 22 76	1:1,79
bitsystem on base 10	1024	0100111001 1011110111 0100011000 0101100010 01100 313 759 280 354	1:1,6
bitsystem on base 11	2048	01001110011 01111011101 00011000010 11000100110 ... 627 989 194 1574	1:1,46
bitsystem on base 12	4096	010011100110 111101110100 011000010110 0010011001 1254 3956 1558 613	1:1,33
bitsystem on base xx	*div*	x x x x x	1: xxx

Lengths ratio gives an idea about length of plaintext in comparison with length of ciphertext. In bit system on base 8 plaintext and ciphertext are of equal length, not so in all other bit systems.

2 „System Alphabet“

The system alphabet is the most important component of computer techniques. It establishes foundation for visualization of bit series content. Without definition of an appropriate alphabet the computer will not be able to work, at all. In bit system on base 8 the fundamental system alphabet is the ASCII-character set in its respective charateristics. For each other bitsystem an own system alphabet has to be defined. System alphabets are all independent from each other.

3 „Bit Conversion“

Bit conversion is changing a series of bits from one bit system to another. Number of bits and their order remain invariable. No bit is added and no bit is removed. Number of bits in one segment (unit) change and by this the structure of the bit series, as well. Decimal values of the new units serve for indexation to the assigned system alphabet.

Bit conversion from base 1 to base 7, 8 and 12 occurs as follows:

bit series base 1:
010011100110111101110100011000010110001001100101011011100110010 . . . . . .
bit series base 7:
0100111 0011011 1101110 1000110 0001011 0001001 1001010 1101110 0110010 . . . . .
Index: 39 27 110 70 11 9 74 110 50
' ESC n F VT TAB J n 2

bit series base 8:
01001110 01101111 01110100 01100001 01100010 01100101 01101110 01100101 . . . .
Index: 78 111 116 97 98 101 110 101
system alphabet base 8:
N o t a b e n e

bit series base 12:
010011100110 111101110100 011000010110 001001100101 011011100110 0101 . . . . .
Index: 1254 3956 1558 613 1766
system alphabet base 12:
&ä }Æ Ó8 vâ ßä

The initial bit series on base 1 is changed by bit conversion to the corresponding units (segments) of the aimed bit system. No bit is added and no bit is removed. The unstructured bit series (number and order of sequences) on principle remain the same. Decimal values of the accruing bit sequences correspond with the index of the cipher assigned in the appropriated sytem alphabet (cipher alphabet).

Up to now conversion of bit sequences are performed in procedure **„Coding Base 64“,** only [#3]. 8-bit sequences are converted to 6-bit segments which each gets a specific cipher character from a 64 elements alphabet (changing from bitsystem on base 8 to bitsystem on base 6). Decimal values of the 6-bit segments constitute index number of the connected system alphabet. The system alphabet is stated fix.

4 Uniform Order System

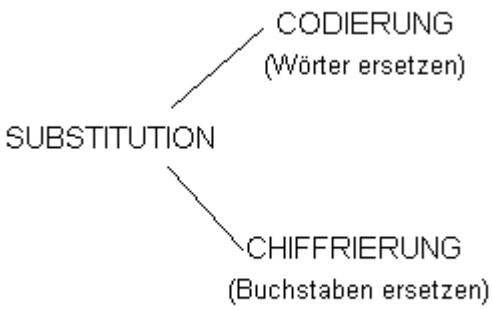
In current cryptography **inputs** and **outputs** (plaintexts, encrypted results, cipher texts) are processed in **bytes**. Further it is demanded plaintext and ciphertext should have the same length [#4,#5,#6]. Ciphertext had to be stored at the same place the plaintext stood before. Thus, for each plaintext character a definite ciphertext character exists. Inputs and outputs are displayed in the same cipher alphabet, in extended ASCII-character set.
Consequently all encryption steps are performed in bitsystem on **base 8** and insofar in an „**uniform order system**“.

By this way, only statistically repetition patterns, word combinations, frequency structures and bigramms in ciphertext can lead to comparable features in plaintext [#7]. Best known attacks like analysing structures, „known plaintext attack“, „chosen plaintext attack“ and even differential and linear crypto analysis will have success only, if there is a definite plaintext character for each ciphertext character, too. Hence, a uniform order system is underlying.

Actual techniques, applied up to now like (Feistel net, Intermix, LFSR, S-boxes, modes EbC, CbC, CFb, confusion and diffusion etc) which demand plaintext and ciphertext having ratio **1:1** are not applicable in CypherMatrix prodedures. It works in all bitsystems from base 1 up to base 16 (exept base 8).

5 „Letter Codification“

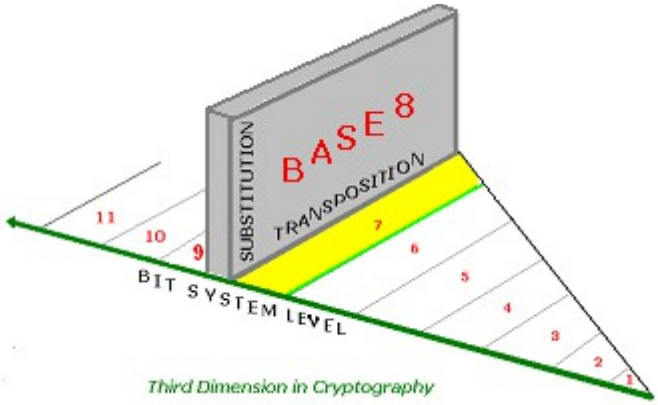
By the time two methods have been developed: „substitution“ and „transposition“ (letter codification). When introducing digital techniques bits have gotten the places of letters but yet the principles of letter codifications were continued . Notwithstanding changeover to digits as a matter of fact still letters are coded. „Verschlüsselt wird nach wie vor gemäß den Grundsätzen der Substitution und Transposition“ [#8].



Simon Singh, Geheime Botschaften, S.48

Letter codification may be now complemented by „**code encrypting**“. Bitsystem on base 8 used up to now has only to be converted into another bitsystem (base 1 up to base 16, exept base 8). The bits represent no letters any more but a digital information in the concerning bitsystem as substitutions for word segments. Above all ciphers cannot be analysed any longer by repetition patterns, word combinations and other analysing attacks, because of absentee connections with letters.

By this appropriation of connotations definite cryptographical operations may be summarized as an own area with special term „**Codegraphy** as part of cryptography“ and treated as a separate part of cryptography. The hitherto used fields „Substitution“ and „Transposition“ are complemented by a third dimension: „Bit System Level“.



With this „third dimension“ a lot of coding solutions can be created. Besides in appendix A some programs are listed developed by the author. For each single program source code may be required from the author per e-mail (eschnoor@multi-matrix.de). Programs are still written in „WindowsXP“, they have to be transcribed to **C#** [#9].

6 CypherMatrix Procedure

CypherMatrix procedure establishes an own field. Following techniques are used: only: startsequence with optimal 42 characters, generating of block keys, matrix keys and system-alphabets as well as system-changing by bit conversion. For instance by changing of characters from bitsystem on base 8 (inputs) to ciphertext in system on base 7 (outputs). All further steps to defence attacks used in current coding procedures are not necessary. The uniform order system on base 8 ceased and both areas plaintext and ciphertext cannot compared meaningful any longer. Thereby bases of all attack scenarios are not applycable and one may forget them.

6.1 Generator

The procedure begins with inserting of a **start sequence** and leads to a unmistakable and collisionfree mapping by way of a **CypherMatrix** (16x16 characters: GF(16^2)). CypherMatrix is incomparable with a „s-box“. CypherMatrix provides the function with all control parameters (cipher alphabet, block key, matrix key as starting sequence for the next run and further parameters) necessary for cryptographic solutions. As to the rules of probability a recurrence of the same CypherMatrix will occur first in **256!** (faculty) = **8E+506** cases.

6.1.1. Course of the Function

The procedure is symmetric because sender and recipient have to insert the same start sequence in order to initiate generator and fundamental system alphabet and it is polyalphabetic because generator creates new control parameters in each round. A round is the pass of the generator from inserting start sequence (esp. matrix key) to generating the CypherMatrix.

The procedure begins with the start sequence and continues as from second round with the first matrix key, which is extracted from the current CypherMatrix with 42 signs. By this an unlimited number of rounds results until an end impulse is set. Any whatever startsequence (passphrase) with at least 36 signs (optimal 42) controls the whole procedure. Some examples:

7 kangaroos jumping along the Times Square	[43 bytes]
Horse racing on the banks of San Sebastian	[42 bytes]
3 koala bears are diving at Murray's Mouth	[42 bytes]
Ein Fliegenpilz steigt in die Stratosphäre	[42 bytes]

The start sequence should be easy to remember and possibly somewhat catchy and of funny words. The chosen phrase cannot be guessed, can easy be kept in mind and may not be written down, anywhere. Because of their length lexical attacks and iterative searching should be impossible. An attacker even isn't able to analyse parts of the start sequence neither apart nor successive because the passphrase could be found in total only, if at all.

Following working parameters control the function:

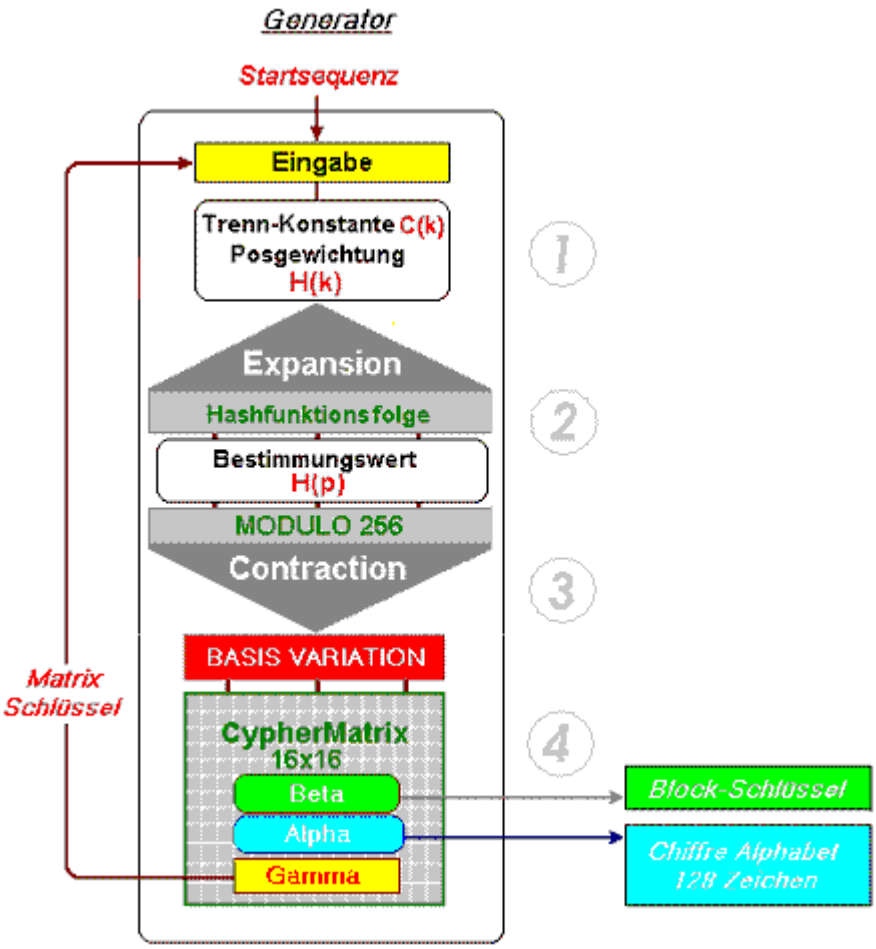
a) personal input:

1) start sequence (36-64,optimal 42),
2) user code (1 up to 999, preempt with 1),

b) constant connected:

1) length matrix key (36-64, preempt with 42),
2) block length (7-63, preeempt as necessary),
3) expansion factor (36-128, preempt with 77).

The start sequence in its function approximates the “Spruchschlüssel” of the **Enigma** and the working parameters meets in its function the “Tagesschlüssel” [#15]. With each round a new cipher alphabet is be generated directly as at Enigma when inserting a letter and proceeding the key roll [#16]. The fundamental difference between both encryptions is set up by the fact that in plaintexts and ciphertexts as well Enigma operates with letters in bitsystem in base 8 while CypherMatrix procedures perform a system change from base 8 to base 7.



In each pass generator creates control parameters necessary for encryption:

1. Cipher alphabet for the current round,
2. block keys for XOR concatenation and
3. matrix keys as start sequence for the next round.

Completion of a round constitutes the **CypherMatrix** with 16x16 elements, which provides the function with all control parameters necessary for encoding. Matrix key is beeing lead back (loop) to the beginning of the function and initializing the next round. By this an unlimited number of rounds arises until an end impulse is set.

6.1.2 Inserting Start Sequence

As an example we insert the following start sequence:

Checkpoint Charly at Madison Square Garden [42 bytes]

43 68 65 63 6B 70 6F 69 6E 74 20 43 68 61 72 6C 79 20 61 74 20
4D 61 64 69 73 6F 6E 20 53 71 75 61 72 65 20 47 61 72 64 65 6E

The goal is to find an evident determination base for analysing the start sequence.

Input **m** is a series of definite bytes **a(i)** with lengths **n.** In order to analyse the series as state of facts the single characters have to be systematized (scaled). For this each byte **a(i)** gets an index and all bytes will be appropriate associated with each other (addition).

$$\mathbf{m} = (a_1+1) + (a_2+1) + (a_3+1) + \ldots (a_i+1) + \ldots (a_n+1)$$

(single value of „a(i)“ has to increase by (+1) because otherwise ASCII-zero (0) would not be considered)

$$\mathbf{m} = \sum_{i=1}^{n} (a_i + 1)$$

$$\mathbf{m} = 3991$$

In order to differentiate single bytes **a(i)** inside the series additional criterions have to be added, because otherwise no definite results will resume.

6.1.3 Extending to Position Weighting

Every object (fact) – which is scalable in its dimensions – can be exactly determined by coordinates for **subject, location** and **time** (cartesian coordinate system). We set:

object: (**m**) digital information of length (**n**)
subject: (a_i) elements of information, signs, bytes
location: (p_i) position of byte **a**(**i**) inside the information
time: (t_i) point of time of byte **a**(**i**) inside the information

In order to distinguish single characters each byte **a**(**i**) is multiplied with its location **p**(**i**). Time will be relevant only if there exists a functionally connection between a single byte and clock frequency. Normaly this connction is constant and we may set: **t = 1**. In order to get an exact determination for series **m** we associate dimension values subject, location and time by multiplying and summarize all results to a destination value **H**(**k**):

$$\mathbf{H(k)} = \sum_{i=1}^{n} (a_i + 1) * p_i * t_i \qquad t_i = 1$$

$$\mathbf{H(k)} = 85589$$

With position weighting the single bytes **a**(i) differ but collisions in consequence of exchanging bytes inside the series are not excluded, yet.

6.1.4 Excluding Collisions

A collision occurs at following conditions:

collision: $\mathbf{H(k)}\ a_i = \mathbf{H(k)}\ b_i$

$$(a_1+1) * p_1 + (a_2+1) * p_2 = (b_1+1) * p_1 + (b_2+1) * p_2$$

To avoid collisions the position weighting is shifted by a distance **C** to an area above length **n,** that means: p_i will be extended by a constant distance **C.**

Further development is demonstrated in article: "Determinants leading to collisionfree"
Result leads to following formula: $\mathbf{C = n * (n - 2)}$

Factor **C** depends on length **n** of the start sequence only. **C** furthermore includes the properties being equal for all start sequences with same lengths and deviding values of position weighting in collisionfree and collision burdened segments. Because of this factor **C** gets the name: dividing constant **C**(**t**).

In order to individualize the function an additional code is introduced – a chosen number between 1 and 99. We set code = 1:

$$\mathbf{C(t) = n * (n - 2) + code}$$
$$\mathbf{C(t) =} \ 1681$$

Each pass of the function is counted with round. By including dividing constant **C**(**k**) the destination value $\mathbf{H_k}$ results as follows:

$$\mathbf{H_k} = \sum_{i=1}^{n} \mathbf{(a_i + 1) * (p_i + C(t) + round)}$$
$$\mathbf{H_k} = 6798451$$

The result **H**(**k**) avoids collisions but is still too narrow to establish anvulnerable destination values. Probably it could serve as **MAC** for messages.

6.1.5 Extending to Hash Function Series

To extend the destination base an **expansion function** (**HF**) is introduced which widens the sequence to an extensive series in a superior number system. The number system of expansion may be chosen between 64 and 96. Here number system on **base 77** is fixed. Any other number in the defined range may be chosen. The function calculates for each value of the inserted sequence its decimal value $\mathbf{s_i}$ which is changed to $\mathbf{d_i}$ **-** digits in number system on base 77. Simultaneous the function calculates the sum of all single results $\mathbf{s_i}$ as an additional destination value **H**(**p)** for creating several control parameters and accumulates the results $\mathbf{d_i}$ serial as hash function series (HF).

$$\mathbf{s_i = (a_i + 1) * p_i * H_k + p_i + code + round}$$
$$\mathbf{s_i \rightarrow d_i} \ \text{(base 77)}$$

$$\mathbf{HF = d_1 + d_2 + d_3 + \ldots + d_i + \ldots d_m}$$
(m = number of digits in system on base 77)

$$\mathbf{H_p = \sum_{i=1}^{n} S_i}$$
$$\mathbf{H_p =} \ 581872623626$$

The chosen number system on base 77 comprises the following digits:

0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz&#@àáâãäåæçèéêë
(determinated by the author, not standardized)

By generating hash function series the sequence “Charly” at position 12 of the inserted passphrase results to the following calculations:

char	pi		Hk	(ai+1)*pi*Hk	Si		Basis 77
	(ai+1)	(ai+1)*pi			pi+code+r		
..	...	.. ...	.....	.....	..	.....	.....
C	68	12 816	6798451	5547536016	14	5547536030	23&YPa
h	105	13 1365	6798451	9279885615	15	9279885630	3WêáxF
a	98	14 1372	6798451	9327474772	16	9327474788	3YQ8LG
r	115	15 1725	6798451	11727327975	17	11727327992	4Pkzeç
l	109	16 1744	6798451	11856498544	18	11856498562	4TLuw&
y	122	17 2074	6798451	14099987374	19	14099987393	5G7æGU
..	...	.. ...	.....	.....	..	.....	.....

The hash function series comprises 248 digits in number system on base 77:

DBlxäelGF4xDyëæ10Rjdo1RXUnp1s9Wbf1élåu929ëéæs2dFjB02çL2HuåFkrk23&YPa3W
êáxF3YQ8LG4Pkzeç4TLuw&5G7æGU1bâfVr4q7æuL5âg4ZX1v1zBN4Náxuã5oåU7P66yçfb
6ocW2n7iLroz7j##&T7&5sãU2V6B8K6PReQW8âZbAá9bKGçC89Y30Z9#DqFw8éWiOb2êw5
#R6rFä3z9RFáZeBKTLTUABOhä9AcyCOLBsl5Pi

The variables are digits (not characters). There is no way back to the start sequence (first **one-way-function**).

Simultaneous the function calculates the following destination Data:

dividing-constant C(t):	1681
position weighted value (H_k):	6798451
destination value (H_p):	581872623626
total value (H_p+H_k):	581879422077

Control parameters derived from destination Data show as follows:

Variante	(H_k MOD 11) +1	=	1	begin of contraction
Alpha	((H_k + H_p) MOD 255) +1	=	148	offset cipher alphabet
Beta	(H_k MOD 169) +1	=	89	offset block key
Gamma	((H_p + code) MOD 196) +1	=	128	offset matrix key
Delta	((H_k + H_p) MOD 155) +code	=	93	dynamic bit series
Theta	(H_k MOD 32) +1	=	20	offset back factor
Omega	(H_k MOD 95)+1	=	62	parameter cipher alphabet
Kappa	(H_k MOD 16384) +1	=	7044	begin of cipher alphabet

The above control parameters serve for solution of several cryptographical tasks.

6.1.6 Contraction to BASIC VARIATION

In order to reduce the variables back to decimal numbers a contract function is introduced. The digits of hash function series are assumed to be digits in number system on **base 78** (expansion base +1). Each three digits of the function series are reconverted in serial manner by **MODULO 256** to decimal numbers **0** to **255** (without repetition). The parameter **Theta** is deducted.

Results are stored in BASIC VARIATION, an array of 16x16 elements. A backwards searching of foregoing Data is not possible (second **one-way-function**).

The first five reconversions beginning at variante = **1** show as follows:

3 digits base 78	decimal	Modulo 256	- Theta	element
DBl	79997	125	20	105
Blx	70649	249	20	229
lxä	290619	59	20	39
xäe	364378	90	20	70
äel	422963	51	20	31

BASIC VARIATION (256 elements)

```
105 229 039 070 031 238 215 194 219 003 074 116 075 191 150 203
083 117 118 005 157 011 196 133 006 251 124 045 034 181 220 192
232 107 044 119 158 140 120 126 066 239 108 190 007 101 135 041
186 090 208 121 122 173 218 071 047 068 161 123 134 195 156 062
217 182 012 136 067 137 175 125 174 127 176 233 089 130 228 128
200 081 226 024 129 097 231 193 076 183 061 250 167 149 252 072
082 087 057 211 213 247 210 236 221 037 035 109 055 201 084 008
111 199 202 184 015 027 085 017 094 216 064 223 166 110 138 180
069 168 177 032 237 234 148 020 245 178 159 073 230 235 198 058
106 197 204 056 088 249 209 240 063 212 152 042 065 131 049 033
132 026 036 160 241 139 242 222 043 046 077 114 227 153 162 038
154 009 010 142 185 091 048 104 100 040 188 151 243 050 224 092
163 187 016 086 205 141 028 014 189 143 244 018 155 051 144 013
025 019 112 164 206 093 113 246 225 145 029 115 169 078 248 253
146 052 165 254 079 255 021 214 171 000 147 002 102 095 170 172
207 179 001 004 022 023 030 053 054 059 060 080 096 098 103 099
```

6.1.7 Calculation of „CypherMatrix“

Calculating CypherMatrix the elements of BASIC VARIATION are used directly in their distribution 16x16 to achieve the destination base. The elements values are related to index values of bytes (**0** to **255:** comparable with ASCII-set). To distribute the characters in the CypherMatrix (16x16) the function used the following determination:

Index values (**i, j**) are generated in an separate array (IndexFolge(2,16)) out of the elements BASIS-VARIATION (VarFolge():

Detail of source code:

```
SUB DynaFolge

  LOCAL DynA(),Z
  SHARED IndexFolge(),VarFolge(),Delta,Omega

  DIM DYNAMIC DynA(16)

  FOR B=1 TO 2
     IF B=1 THEN
        Z=Delta
     ELSEIF B=2 THEN
        Z=Omega
     END IF

     FOR C=1 TO 16
        INCR Z
        IF Z>256 THEN Z=1
        A=(VarFolge(Z) MOD 16)
        DynA(C)=A
        IF C>1 THEN
          L=0
          DO
             INCR L
             IF DynA(L)=A THEN
                INCR A
                A=(A MOD 16)
                DynA(C)=A
                L=0
             END IF
          LOOP UNTIL L=C-1
        END IF
        IndexFolge(B,C)=DynA(C)+1
     NEXT C
  NEXT B

  ERASE DynA
  END SUB

   N = 0
   FOR i = 1 TO 16
      FOR j = 1 TO 16
         INCR N


         x = IndexFolge(1,i)
         y = IndexFolge(2,j)
         Matrix$(x,y)= Variation$(N)          CypherMatrix Array
      NEXT j
   NEXT i
```

The calculation results in following two index series:

```
Index-Folge 1:  6 13 9 3 8 10 4 7 11 5 14 15 12 16 1 2
Index-Folge 2:  13 15 10 7 14 9 4 11 16 1 2 3 5 12 6 8
```

The final CypherMatrix comes out as follows:

Final CypherMatrix

1	ª	n	è	Ï	┤	¿	U	▒	^	←	@	E	©	◄	☼	■	*16*
17	µ	Ù	ã	▓	:	┼	ö	╠	§	Û	ƒ	j	□	¶	Ý	I	*32*
33	ø	3	É	Å	♪	‼	∟	p	¢	ì	⌠	↓	V	♬	═	↕	*48*
49	`	b	g	;	c	Õ	▲	'	6	↨	<	i	♦	5	▄	P	*64*
65	▪	e	ç	´	)	Z	x	ð	B	î	l	╟	w	~	×	¥	*80*
81	A	â	1	È	!	→	Ð	$	?	¨	ÿ	ä	8	≡	X	*	*96*
97	K	┐	û	♥	╦	u	Î	v	▉	¯	J	S	F	┬	▼	t	*112*
113	®	N	°	æ	²	4	q	Ñ	ß	]	↔	Æ	ñ	÷	╬	s	*128*
129	¾	2	Ó	(	\	╗	0	▶	d	[	╝	ú	Ä	h	╣	ù	*144*
145	f	_	¬	□	¼	┃	§	☺	½	▯	ô	¤	■	Í	O	☻	*160*
161	"	Ă	▄	¹	┗	k	─	,	♠	♂	|	Þ	♣	à	Ø	-	*176*
177	°	ò	³	À	H	W	þ	9	L	a	=	R	↑	┻	ü	·	*192*
193	Ò	Ö	ó	.	&	°	═	◙	+	ï	M	Ü	á	Ì	±	r	*208*
209	å	├	£	D	>	Â	┌	♀	/	¡	í	┘	y	G	z	{	*224*
225	Y	é	õ	▯	Ç	Q	»	Ô	«	ë	▒	╚	ê	}	C	Ú	*240*
241	7	╔	T	%	◘	Ã	Ê	╩	¦	¸	#	o	Ë	ý	€	m	*256*

CypherMatrix (hex)

1	A6	6E	8A	D8	B4	A8	55	B1	5E	1B	40	45	B8	11	0F	DF	*16*
17	E6	EB	C6	B2	3A	C5	94	CC	F5	EA	9F	6A	20	14	ED	49	*32*
33	9B	33	90	8F	0D	13	1C	70	BD	8D	F4	19	56	0E	CD	12	*48*
49	60	62	67	3B	63	E5	1E	27	36	17	3C	69	04	35	16	50	*64*
65	07	65	87	EF	29	5A	78	D0	42	8C	6C	BA	77	7E	9E	BE	*80*
81	41	83	31	D4	21	1A	D1	24	3F	F9	98	84	38	F0	58	2A	*96*
97	4B	BF	96	03	CB	75	D7	76	DB	EE	4A	53	46	C2	1F	74	*112*
113	A9	4E	F8	91	FD	34	71	A5	E1	5D	1D	92	A4	F6	CE	73	*128*
129	F3	32	E0	28	5C	BB	30	10	64	5B	BC	A3	8E	68	B9	97	*144*
145	66	5F	AA	00	AC	B3	15	01	AB	FF	93	CF	FE	D6	4F	02	*160*
161	22	B5	DC	FB	C0	6B	C4	2C	06	0B	7C	E8	05	85	9D	2D	*176*
177	A7	95	FC	B7	48	57	E7	39	4C	61	3D	52	18	C1	81	FA	*192*
193	E3	99	A2	2E	26	09	F2	0A	2B	8B	4D	9A	A0	DE	F1	72	*208*
209	86	C3	9C	44	3E	B6	DA	0C	2F	AD	A1	D9	79	47	7A	7B	*224*
225	59	82	E4	7F	80	51	AF	E2	AE	89	B0	C8	88	7D	43	E9	*240*
241	37	C9	54	25	08	C7	D2	CA	DD	F7	23	6F	D3	EC	D5	6D	*256*

6.1.8 Control Parameter

As cipher alphabet (system alphabet of bitsystem on base 7) 128 characters are taken from the actual CypherMatrix at position **148.** Certain signs (hex: 00 bis 20, 22, 2C, B0, B1, B2, D5, DB, DC, DD, DE, DF and others) are ignored because they do still their original task (e.g. **1A** = ASCII-26) and disturb the proper realization of the program.

Cipher Alphabet (base 7)

1	¼	│	½	□	ô	¤	■	Í	O	Á	¹	└	k	─	|	Þ	*16*
17	à	Ø	-	°	ò	³	À	H	W	þ	9	L	a	=	R	┴	*32*
33	ü	·	Ò	Ö	ó	.	&	═	+	ï	M	Ü	á	±	r	å	*48*
49	├	£	D	>	Â	┌	/	¡	í	┘	y	G	z	{	Y	é	*64*
65	õ	□	Ç	Q	»	Ô	«	ë	╚	ê	}	C	Ú	7	╔	T	*80*
81	%	Ã	Ê	╩	¸	#	o	Ë	ý	m	ª	n	è	Ï	┤	¿	*96*
97	U	^	@	E	©	µ	Ù	ã	:	┼	ö	╠	§	Û	ƒ	j	*112*
113	Ý	I	ø	3	É	Å	p	¢	ì	▕	V	═	`	b	g	;	*128*

Cipher Alphabet (hex)

```
AC B3 AB FF 93 CF FE D6 4F B5 FB C0 6B C4 7C E8
85 9D 2D A7 95 FC B7 48 57 E7 39 4C 61 3D 52 C1
81 FA E3 99 A2 2E 26 F2 2B 8B 4D 9A A0 F1 72 86
C3 9C 44 3E B6 DA 2F AD A1 D9 79 47 7A 7B 59 82
E4 7F 80 51 AF E2 AE 89 C8 88 7D 43 E9 37 C9 54
25 C7 D2 CA F7 23 6F D3 EC 6D A6 6E 8A D8 B4 A8
55 5E 40 45 B8 E6 EB C6 3A C5 94 CC F5 EA 9F 6A
ED 49 9B 33 90 8F 70 BD 8D F4 56 CD 60 62 67 3B
```

Block Key

The block key drawn from the CypherMatrix as from position **89** comprises 42 signs:

?¨ÿä8 X*K┐û♥╦uÎv▀¯JSF┬▼t®N°æ²4qÑß]↔Æñ÷╬s¾2

```
3F F9 98 84 38 F0 58 2A 4B BF 96 03 CB 75 D7 76 DB EE 4A 53 46
C2 1F 74 A9 4E F8 91 FD 34 71 A5 E1 5D 1D 92 A4 F6 CE 73 F3 32
```

Matrix Key

As start sequence for the next round the function draws from the CypherMatrix as from position **128** a next matrix key with 42 signs:

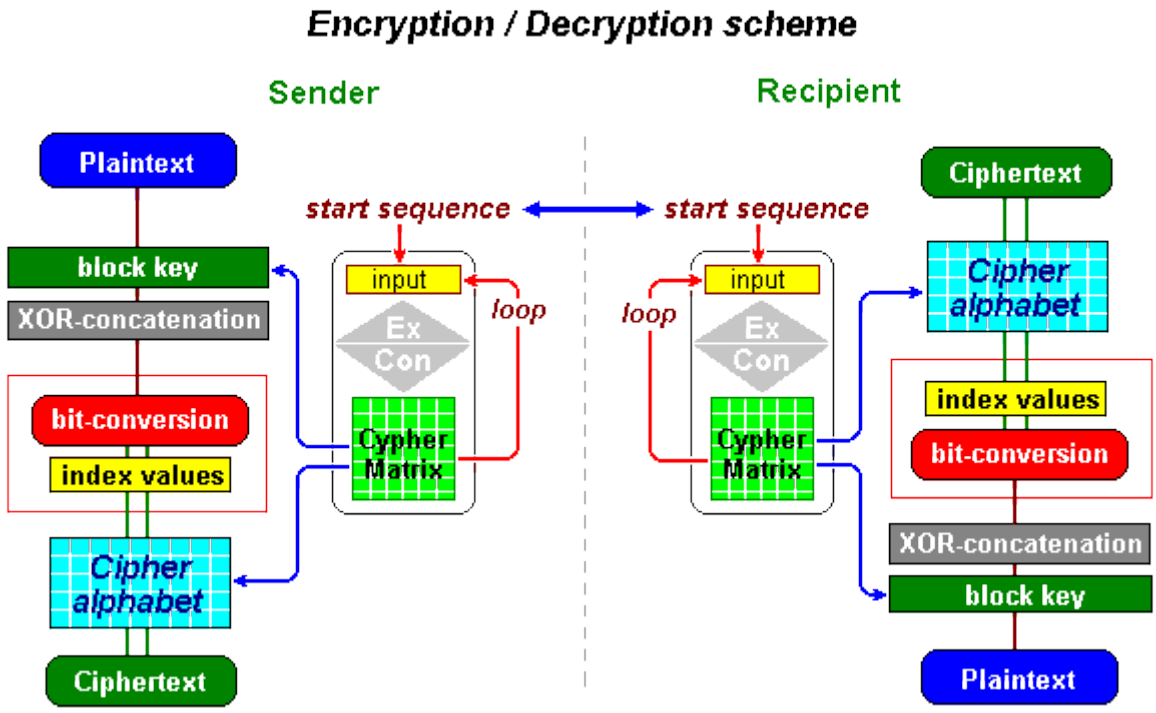
s¾2Ó(\╗0►d[╝úÄh╣ùf_¬¼│§©½□ô¤■ÍO☻"Á▄¹└k─,♠

The matrix key is lead back to the beginning of the function (**loop**). The respective matrix key controls the total operation of the function, of equal volume at sender and at recipient, as well.

7 Coding Area

Coding – writing and reading of secret informations by peculiar methods – takes place in coding area, exclusively. By inserting an identical start sequence at sender and recipient as well an equal course and identic control parameters are generated. As from second round the matrix-key links the course back to begin of the function. by this way a unlimited number of rounds is established until an end instruction is set.

The following scheme shows the connections:



Encryption is performed by following alternatives:

1. **Basic coding:** bit conversion without further operations or
2. **Compound coding:** bit conversion with additional operations ,

   a) with XOR-concatenation (foregoing or succeding) or
   b) connected with further operations (dyn24, exchange).

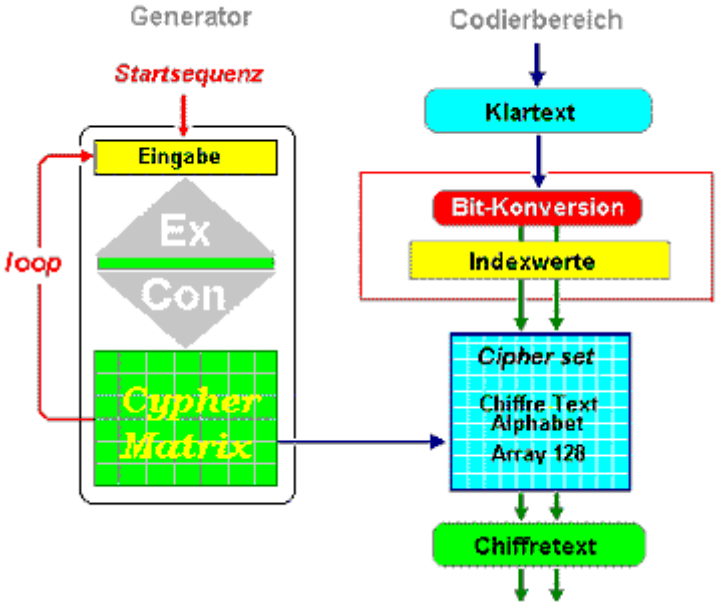
7.1 „Basic coding“

In „basic coding“ mode bit conversion is changed directly from previously plaintext in bit system on **base 8** to the ciphertext in the aimed bit system (base 2 up to base 14, except base 8). From bit series of the inserted plaintext block of 42 x 8 bits (336 bits) results in bitsystem on base 7 a bitseries of 48 x 7 bits (336 bits). Units in bitsystem on base 7 constitute indexes of cipher signs in the appropriate system alphabet. This means the simpliest way of digital encryption. Only the generator is extra necessary. For decoding the indexes get the cipher signs from the respective system alphabet and combine them to the plaintext.

Basic coding performs as follows:

1. Bit conversion
   8-bit plaintext values → 12 bit index values (0 ...4096)
2. Destination of ciphertext
   12-bit Index values → cipher alphabet (0...4096) → ciphertext.

The connections are shown by following scheme:



7.1.1 Demonstrating Basic Coding

Working mode is demonstrated by following program: **System12.exe** (see appendix A):

The inserting start sequence reads as follows:

Donkey racing on the banks of San Bernadino [43 bytes]

Generator calculates following destination Data:

dividing-constant C(t):	1764
position weighted value (H_k):	7142249
destination value (H_p):	628610762269
total value (H_p+H_k):	628617904518

Control parameters derived from destination Data show as follows:

Variante	(H_k MOD 11) +1	=	5	begin of contraction
Alpha	((H_k + H_p) MOD 255) +1	=	4	offset cipher alphabet
Gamma	((H_p + code) MOD 196) +1	=	135	offset matrix key
Delta	((H_k + H_p) MOD 155) +code	=	154	dynamic bit series
Theta	(H_k MOD 32) +1	=	10	offset back factor
Kappa	(H_k MOD 12286) +1	=	8981	begin of cipher alphabet

As an example of **Basic Coding** a text from *Mark Twain* is encrypted:

> *In German, all the Nouns begin with a capital letter. Now that is a good idea; and a good idea, in this language, is necessarily conspicuous from its lonesomeness. I consider this capitalizing of nouns a good idea, because by reason of it you almost able to tell a noun the minute you see it. You fall into error occasionally, because you mistake the name of a*

person for the name of a thing. and waste a good deal of time trying to dig a meaning out of it. German names almost always do mean something, and this helps to deceive the student.

The Awful German Language, Mark Twain, A Tramp Abroad, 1880

As first plaintext block 36 bytes are inserted:

In German, all the Nouns begin with□

49 6E 20 47 65 72 6D 61 6E 2C 20 61 6C 6C 20 74 68 65
20 4E 6F 75 6E 73 20 62 65 67 69 6E 20 77 69 74 68 20

Plaintest in bitsystem on **base 8** (36x8=288) is converted into segmemts of aimed bitsystem on **base 12** (288:12=24). Decimal values of these segments (base 12) are index values of character positions in the cipher alphabet (system alphabet). Indexes have to be added by +1, because index „0“ will not be identified at the array of cipher alphabet.

Plaintext base 8:

	I	n		G	e	r	m	a
01001001 01101110 00100000 01000111 01100101 01110010 01101101 01100001 . . .								
base 12:								
010010010110 111000100000 010001110110 010101110010 011011010110 0001 . . .								
Index:	1174		3616		1142		1394	1750
(+1)	1175		3617		1143		1395	1751
ciphertext:	îi		Žs		îC		{8	}˜

system alphabet base 12:

Alphabet$(1143) = îC
Alphabet$(1175) = îi
Alphabet$(1395) = {8
Alphabet$(1751) = }˜
Alphabet$(3617) = Žs

Ciphertext reads as follows:

îiŽsîC{8}˜è4ê„å£}ˆŠs€ˆ,§éQÅ5€ÿÅ9éSé§}C…4éT€¬€ˆ,sjLX’jQX˜lèZ›b9v”
léf”lbzZg•#¦bAf—jQfZkQdZjJxDj¥#ž,êéž,î€–{xé–ƒ›|f,Réœƒ¬”™íFˆ™,‹ër
íFˆ£íG| ƒWïfƒîë£c¦a c¥alU9i¢U9s”cva¢evS¡dPo©U9Wžd–WŸdPW¤d¦a¢R}k•
X‘oiNäcpX KhXrmaX¡oiWLmaX¢P™Nâb‹V¡ojX¡cZWMN‹Y5ae•’ˆN“êˆT•¦ŠW”êœT

```
EE 69 8E 73 EE 43 7B 38 7D 98 E8 34 EA 84 E5 A3 7D 88 8A 73 80
88 82 A7 E9 51 8F 35 80 98 8F 39 E9 53 E9 A7 7D 43 85 34 E9 54
80 AC 80 88 82 73 6A 4C 58 92 6A 51 58 98 6C E8 5A 9B 62 39 76
94 0D 0A 6C E9 66 94 6C 62 7A 5A 67 95 23 A6 62 41 66 97 6A 51
66 5A 6B 51 64 5A 6A 4A 78 44 6A A5 23 9E 82 EA E9 9E 82 EE 80
96 7B 78 E9 96 83 9B 7C 66 82 52 E9 9C 83 AC 94 99 ED .. .. ..
```

The encrypted file **Language.ctx** comprises 816 characters. Through bit conversion 12 characters in bitsystem on base 8 comprise 96 bits which will be assigned to 8 double signs in bitsystem on base 12. Insofar there is a ratio of 1:1,333.
The system alphabet on base 12 needs 4096 signs, which are not available by single characters. So, they are generated as double signs by digits of number system on base 128 – that are 16384 digits.

Partial sourcecode:

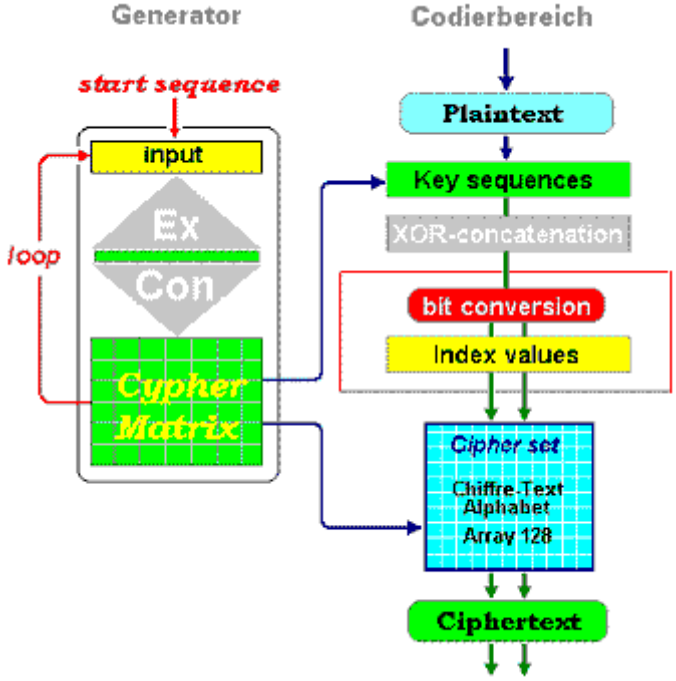
```
SUB Alphabet
    SHARED Alphabet$(),  Kappa
    Kappa = 8981, 1202, 142, 8330, 1428  (anew generated in each round)

    FOR C=1 TO 4096
        X## = C + Kappa
        CALL DezNachSystem (128, X##, Zeichen$)
        Digit$ = „00“+Zeichen$
        Digit$ = RIGHT$(Digit$,2)
        Alphabet$(C) = Digit$
    NEXT C
END SUB
```

„DezNachSystem“ is a function to change numbers (X##) from decimal values to digits on base 128 (Zeichen$). Example: 1395 → {8.

7.2 „Compound coding“

By „Compound coding“ several operations are combined in succession with bit conversion and further operations (e.g. XOR concatenation, dyn24, exchange).
If XOR concatenation is installed before bit conversion then encryption works in three steps:

1. partial dynamic „one-time-pad“
   plaintextblock → block key → 8-bit XOR concatenation
2. bit conversion
   8-bit XOR concatenation → 13 bit index values (0 ...8192)
3. destination of ciphertext
   13-bit index values → cipher alphabet (0...8192) → ciphertext.



7.2.1 Dynamic „one-time-pad“

The following example demonstrates encryption steps of file **Asulike.txt** with program **MyCode13.exe** (see appendix A).

As start sequence serves:

Yellowstoneparkbridge is under construction“ [43 bytes]

The file **Asulike.txt** to be encrypted reads as follows:

Now, my co-mates and brothers in exile,
Hath not old custom made this life more sweet
Than that of painted pomp? Are not these woods
More free from peril than the envious court?
Here feel we but the penalty of Adam,
The seasons' difference, as the icy fang
And churlish chiding of the winter's wind,
Which, when it bites and blows upon my body,
Even till I shrink with cold, I smile and say
'This is no flattery: these are counsellors
That feelingly persuade me what I am.'
Sweet are the uses of adversity,
Which, like the toad, ugly and venomous,
Wears yet a precious jewel in his head;
And this our life exempt from public haunt
Finds tongues in trees, books in the running brooks,
Sermons in stones and good in every thing.
I would not change it.

Shakespeare, "As You Like It", Dateien/shake.CSS

Program creates the following destination Data:

dividing-constant (C(t) =	1764
position weighted value (Hk) =	7972450
destination value (Hp) =	780933368332
total value (Hk+Hp) =	780941340782

Variante	(H_k MOD 11) +1	=	3	begin of contraction
Alpha	((H_k + H_p) MOD 255) +1	=	228	offset cipher alphabet
Beta	(H_k MOD 169)+1	=	45	offset block key
Gamma	((H_p + code) MOD 196) +1	=	14	offset matrix key
Delta	((H_k + H_p) MOD 155) +code	=	133	dynamic bit series
Theta	(H_k MOD 32) +1	=	3	offset back factor
Omega	(H_k MOD 95)+1	=	35	parameter cipher alphabet
Kappa	(H_k MOD 12286) +1	=	513	begin of cipher alphabet

The final CypherMatrix is calculated as follows:

CypherMatrix

1	E6	10	17	68	9B	DB	9D	BF	C0	49	11	3B	C8	DC	1D	DD	*16*
17	18	1E	DE	E2	23	2B	4A	55	56	59	5A	5B	5C	01	76	81	*32*
33	CC	72	87	4E	FD	EF	74	4C	CA	B5	3F	70	65	B8	CB	CD	*48*
49	A8	40	66	2A	D2	24	1B	A4	4B	91	F6	C9	77	5F	9F	30	*64*
65	A5	A0	6B	8D	33	A7	4D	03	75	AE	AB	F0	41	F3	C5	2D	*80*
81	29	58	BC	32	B6	67	34	25	88	DF	F7	57	D5	6E	EA	D9	*96*
97	A1	7E	16	C1	2E	EB	06	53	42	26	B1	45	47	B9	6A	04	*112*
113	3A	A6	CE	D6	31	4F	05	39	5E	50	3C	37	AA	CF	60	02	*128*
129	13	D8	F5	EC	51	6F	AC	7D	7F	93	5D	8E	46	0B	6C	6D	*144*
145	71	C2	AF	79	C4	73	A9	54	FE	20	27	E7	ED	FA	D3	BA	*160*
161	69	B7	36	E8	21	8F	78	43	7A	7B	61	19	52	F2	0C	7C	*176*
177	A2	D0	80	82	62	1F	E0	12	B0	9E	83	14	F8	96	3E	84	*192*
193	07	C6	1A	08	22	E9	E5	F9	D1	63	9C	28	AD	92	BB	85	*208*
209	86	C3	F1	89	99	35	C7	E1	BD	8A	E4	EE	A3	1C	B2	8B	*224*
225	2C	09	3D	97	BE	8C	D4	90	64	15	F4	E3	B3	2F	44	94	*240*
241	FB	FC	D7	0D	0A	95	98	0E	0F	DA	FF	00	9A	48	38	B4	*256*

In each round a plaintext block of 39 bytes (39x8 = 312 bits) is XOR concatenated with a block key of equal length (312 bits). 39 bytes are chosen because their length must be divisible by 13. As first plaintext block the program reads:

Now, my co-mates and brothers in exile, [39 bytes]

4E 6F 77 2C 20 6D 79 20 63 6F 2D 6D 61 74 65 73 20 61 6E 64
20 62 72 6F 74 68 65 72 73 20 69 6E 20 65 78 69 6C 65 2C

At position 45 the block key is extracted from current CypherMatrix with 39 bytes:

e¸ËÍ¨@f*Ò$←¤K‘öÉw_Ÿ0¥ kì3§M♥u®«ðAóÅ-)X¼

65 B8 CB CD A8 40 66 2A D2 24 1B A4 4B 91 F6 C9 77 5F 9F 30
A5 A0 6B 8D 33 A7 4D 03 75 AE AB F0 41 F3 C5 2D 29 58 BC

plaintext base 8:
01001110 01101111 01110111 00101100 00100000 01101101 01111001 00100000 . . .
block key base 8:
01100101 10111000 11001011 11001101 10101000 01000000 01100110 00101010. . .
XOR concatenation base 8:
00101011 11010111 10111100 11100001 10001000 00101101 00011111 00001010 . . .

dez:	43	215	188	225	136	45	31	10
hex:	2B	D7	BC	E1	88	2D	1F	0A

+×¼áˆ-▼ ±K6É*å“ºW>ñT…Â↓âGÏ(q♣ŽÂža–½DE=É

2B D7 BC E1 88 2D 1F 0A B1 4B 36 C9 2A E5 93 BA 57 3E F1 54
85 C2 19 E2 47 CF 28 71 06 8E C2 9E 61 96 BD 44 45 3D 90

As result a partial dynamic „**one-time-pad**“ arises [#13]. Plaintext and block key are of equal length and the key will not be repeated. Each round gets another key from the respective CypherMatrix. Because in each round occurs the same prodecure thus a chain of „one-time-pads“ arises which in its effect as a uniform function encodes the whole plaintext. By this way the whole course will become unbreakable [#13].

7.2.2 Bit Conversion

The result of XOR-concatenation in bitsystem on base 8 (39x8 = 312 bits) is converted to characters of bitsystem on base 13 (312 bits = 24x13) . Decimal values of the changed signs (base 13) are index values to positions of cipher characters in the system alphabet on base 13 (cipher alphabet of 8192 elements). Indexes have to be added with (+1) because the cipher alphabet array does not recognize index „**0**“

XOR concatenation base 8:
00101011 11010111 10111100 11100001 10001000 00101101 00011111 00001010 1. . .
conversion base 13:
0010101111010 1111011110011 1000011000100 0001011010001 1111000010101. . .

index:	1402	7923	4292	721	7701
(+1)	1403	7924	4293	722	7702
cipher:	E©	à¡	bå	9}	@N

System alphabet on base 13

. . . . . .
Alphabet$(722) = 9}
Alphabet$(1403) = E©
Alphabet$(4293) = bå
Alphabet$(7702) = @N
Alphabet$(7924) = à¡
. . . . . .

Encryption of plaintext file *Asulike.txt* results in ciphertext file *Asulike.ctx* with 1056 characters, as follows:

Ciphertext (base 13)

E©à¡bå9}@NS€tuMizqX—RXYNMDb8YKJ{EGCSIÅN”AlY}8h#Ik6çAc•QX@ëå
‰plnFsüYOêQáüìœ€ªZŽiäé—ãNâƒk1Vë|
6njîDRžzãPuïGa˜c›hmR¥jKo9ìœWçMrxWZxêšêQ#Å&žê€îJyHj›âsT”KFåbXäMæzRL¡
ob#ac„qIeäé§z3@æí@éx{Orë|gbéMƒvlXreàt–išu aŒUîv‹iVrlEaäokNâye«VááSk,X£zäZr
y¥#‘Z¦uanèäƒrwo1nMXEw™rqdgé@tâRìíOé—xái“@GTrâˆm{ëìïzUIí”“•#œ’2yZŠá…1
å¥–•ê›‰0—ü|ü“§ê˜‹Qà£›ˆææ|øì£é‘âìÅ1îv,ï&2‘ªäY’‡nˆ@gæ‘„’èîkìlš{œæK|nàø†€|
kt‰“Rr¥l‡}l„PìLüY‡„†wV¬l7ërƒ#‰•#lv|k€y‘dBy†ï«çP‡8zulçví…. . . . . .

Ciphertext (hex)

```
45 A9 E0 A1 62 E5 39 7D 40 4E 53 80 74 75 4D 69 7A 71 58 97 52 58 59 4E 4D
44 62 38 59 4B 4A 7B 45 47 43 53 49 8F 4E 94 41 6C 59 7D 38 68 23 49 6B 36
E7 41 63 95 51 58 40 EB E5 89 70 6C 6E 46 73 81 59 4F EA 51 E1 81 EC 9C 80
```

```
AA 5A 8E 69 E4 E9 97 E3 4E E2 83 6B 31 56 EB 7C 36 6E 6A EE 44 52 9E 7A E3
50 75 EF 47 61 98 63 9B 68 6D 52 A5 6A 4B 6F 39 EC 9C 57 E7 4D 72 78 57 5A
78 EA 9A EA 51 23 8F 26 9E EA 80 EE 4A 79 48 6A 9B E2 73 54 94 4B 46 E5 62
58 E4 4D E6 7A 52 4C A1 6F 62 23 61 63 84 71 49 65 E4 E9 A7 7A 33 40 E6 ED
40 E9 78 7B 4F 72 EB 7C 67 62 E9 4D 83 76 6C 58 72 65 E0 74 96 69 9A 75 A0
61 8C 55 EE 76 8B 69 56 72 6C 45 61 E4 6F 6B 4E E2 79 65 AB 56 E1 E1 53 6B
82 58 A3 7A E4 5A 72 79 A5 23 91 5A A6 75 61 6E E8 E4 83 72 77 6F 31 6E 4D
58 45 77 99 72 71 64 67 E9 40 74 E2 52 8D ED 4F E9 97 78 E1 69 93 40 47 54
72 E4 88 6D 7B EB EC EF 7A 55 49 ED 94 93 95 23 9C 92 32 79 5A 8A E1 85 31
E5 A5 96 95 EA 9B 89 30 97 81 7C 81 93 A7 EA 98 8B 51 E0 A3 9B 88 E6 E6 7C
9D EC A3 E9 91 E3 EC 8F 31 EE 76 82 EF 26 32 91 AA E4 59 92 87 6E 88 40 67
E6 91 84 92 E8 EE 6B 8D 6C 9A 7B 9C E6 4B 7C 6E E0 9D 86 80 7C 6B 74 89 93
52 72 A5 6C 87 7D 6C 84 50 EC 4C 81 59 87 84 86 77 56 AC 6C 37 EB 72 83 23
89 95 23 6C 76 7C 6B 80 79 91 64 42 79 86 EF AB E7 50 87 38 7A 75 6C E7 76
. . . . . .
```

The encrypted file *Asulike.ctx* comprises 824 characters. Through bit conversion 39 characters in bitsystem on base 8 comprise 312 bits which will be assigned to 24 double signs in bitsystem on base 13. Insofar there is a ratio of 39:48 = 1:1,23.

The system alphabet on base 13 needs 8192 signs, which are not available by single characters. Hence, they are generated as double signs by digits of number system on base 128 – that are 16384 digits.
Partial sourcecode:

```
SUB Alphabet
    SHARED Alphabet$(),  Kappa
    Kappa = 513, 2942, 2341, 2494, 1808  (anew generated in each round)

    FOR C=1 TO 8192
        X## = C + Kappa
        CALL DezNachSystem (128, X##, Zeichen$)
        Digit$ = „00“+Zeichen$
        Digit$ = RIGHT$(Digit$,2)
        Alphabet$(C) = Digit$
    NEXT C
END SUb
```

„DezNachSystem“ is a function to change numbers (X##) from decimal values to digits on base 128 (Zeichen$). Example: 7702 → @N.

8 Deciphering

For deciphering the generator performes an identic course equal to the encryption process. Deciphering is performed in the coding area, but in reverse order, only.

1. Analysing the ciphertext
   ciphertext → cipher alphabet (0...8192) → 13 bit index values
2. bit conversion
   13 bit index values (0 ...8192) → 8-bit XOR concatenation
3. XOR concatenation
   8-bit XOR concatenation → block key → plaintext block

Program searches by cipher blocks of 48 characters – **24** units in base 13 (312 bits) - the decimal index values of single units in an identical created cipher alphabet (system alphabet) and connects them to a series of 312 bits. This series will be devided into **39** 8-bit sequences (312 bits) in bitsystem on base 8 and concatenated with the respective block key. The original plaintext becomes visible.

As an example deciphering the ciphertext file **Asulike.ctx***:*
The program reads at first the ciphertext block with 48 characters:

E©à¡bå9}@NS€tuMizqX—RXYNMDb8YKJ{EGCSIÅN”AlY}8h#I

cipher:	E©	à¡	bå	9}	@N
index	1403	7924	4293	722	7702
(-1) :	1402	7923	4292	721	7701

0010101111010 1111011110011 1000011000100 0001011010001 1111000010101. . .
conversion to base 8:
00101011 11010111 10111100 11100001 10001000 00101101 00011111 00001010 1. . .
XOR block key base 8:
01100101 10111000 11001011 11001101 10101000 01000000 01100110 00101010. . .
plaintext base 8:
01001110 01101111 01110111 00101100 00100000 01101101 01111001 00100000 . . .
index base 8:

78	111	119	44	32	109	121	32
N	o	w	,	□	m	y	□

The original plaintext appears: **Now, my** co-mates and brothers in exile,

9 Security of the Procedure

To the best known attacks belong analysing structures, „known plaintext attack“ and „chosen plaintext attack“, possibly „differential“ and „linear“ analysis, too. By this attacks it is intended to find conspicuous connections in order to discover a way to the plaintext, possibly. To noticeable events of any language belong statistically repetition patterns and word combinations, frequency structures and two-digit-groups and bigramms [#10].

To analyse this features it is necessary that ciphertext and plaintext share certain structures which can be compared really. Therefore a unit order system must exist which works in plaintext and ciphertext as well. In CypherMatrix this is not the case.

Most in all procedures plaintext and ciphertext have the same length. For each plaintext character there is a definite cipher character. Both areas – inputs and outputs – work in the same system-alphabet (ASCII-character set) and by this in the same uniform **order system**.

A system changing does not take place with the sequence plaintext and ciphertext cannot be compared any longer. Due to changing characters from bitsystem on base 8 to bitsystem 7 upon each plaintext character fall 1,143 ciphertext characters. There is no meaningful base for all traditional attacks scenario. They are without effect and we may forget them.

The s**tart sequence** (passphrase) to initialize the generator may be defined between 36 and 64 bytes (optimal 42 bytes). By inserting per keyboard with abount 100 characters the start sequence comprises following key space:

36 bytes →	100^36 = 1E+ 72	length: 240 bit	entropie: 239.188
42 bytes →	100^42 = 1E+ 84	length: 280 bit	entropie: 279.042
64 bytes →	100^64 = 1E+128	length: 426 bit	entropie: 425.207

Start sequence is used only once to start the generator and will not be stored on hard disk.

The **matrix key** extracted from CypherMatrix (256 elements) with 42 characters develops maximal following key space:

42 bytes → 256^42 = 1.4E+101 length: 336 bit entropie: 336.000

By an attack on matrix key with 42 byte length results an entropie of 336 and a exponentielle Komplexität of O(2^336) = **1.4E+101** [#11]. Assumed in a brute force attack an exhausting round takes 1 nano second then 5E+89 years are necessary to find the matrix key reliably. That will take a long time.

Most important steps necessary for the security above all are:

a) avoiding collisions,
b) first „one-way-function“: „expansion“ to hash function series and
c) second „one-way-function“: „contraction“ to BASIC VARIATION.

Embeding this functions into backwards directed destination of foregoing Data is not possible.

CypherMatrix is generated anew in each round. A repetition will occur first in 256! (faculty) = 8.5E+506 cases (2*10^1684).

System alphabet with 128 elements take 212 bytes out of the CypherMatrix (256 elements). 44 elements will remain unnoticed.

212 bytes → 212^128 = 5.9E+297 range: 989 bit entropie: 989,173

9.1 „Compound Coding“

By „Compound coding“ several operations are combined in succession with bit conversion and further operations (e.g. XOR concatenation, dyn24, exchange).
If XOR concatenation is installed before bit conversion then encryption works for instance in three steps:

1. partial dynamic „one-time-pad“
   plaintextblock → block key → 8-bit XOR concatenation
2. bit conversion
   8-bit XOR concatenation → 7 bit index values (0 ...127)
3. destination of ciphertext
   7-bit index values → cipher alphabet (0...127) → ciphertext.

At „combined coding“ there exists a partial „**one-time-pad**“ for each plaintext block [#12]. Plaintext block and block key are of equal length and the key will not be repeated. In each round another key is extracted from the regarding CypherMatrix. To the whole procedure this results in a chain like a connected „one-time-pad“ function. For nowadays understanding this means an absolute security [#13].

9.2 „brute force“ Attack

There is still the possibility to break the cipher by a „brute force attack“ [#14]. An attacker in principle knows the ciphertext and the CypherMatrix procedure. but he does not know the respective program and single control parameters, including start sequence. He has the following offensive means, only:

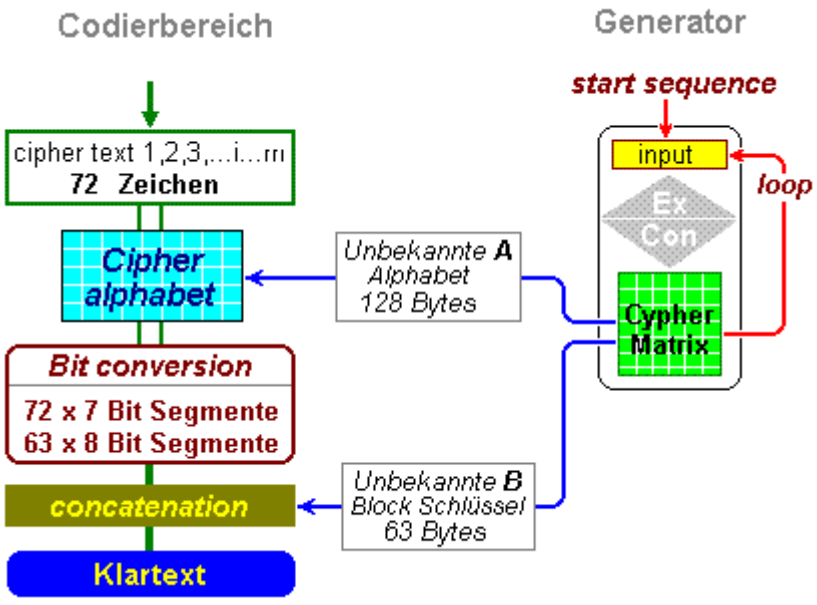
1. Searching startsequence at beginning of the procedure,
2. getting into the running program or
3. finding the way from ciphertext to plaintext backwards from end point.

Key space necessary for an attack on the start sequence is already discussed above.

An attempt to infer from the running program on start sequence or matrix key is hopeless because between both areas there exists no interconnection . Start sequence and matrix keys are not used for encryption steps, at all.

Further an attempt remains to rely upon the ciphertext and roll up the course from the very end. For this it will be necessary to find out the system-alphabet on base 7 and the distribution of all 128 elements. This distribution contains a combination space of 128! (faculty) = **2.85E+215** possibilities. The correct distribution may be searched by an iteration test. But at those volume it will be beyond all hope.

If for a definite round the system alphabet on base 7 should be known for wich reason whatsoever the problem comes up to find the block key. Decryption in each round happens as follows:



The procedure includes three functions:

1. Plaintext block --> **block key** --> -8 bit XOR sequences
2. 8-bit XOR sequences --> 7-bit index values
3. 7-bit index values --> **chipher alphabet (128)** --> ciphertext

In this functions the parameters **block key** and **cipher alphabet** are two variables independent from each other. Effective are:

cm = f [f1 (an, **k1**), f2 (b1, b2), f3 (b2, **k2**)]
an = f [f3 (cm, **k2**), f2 (b2, b1), f1 (b1, **k1**)]

fx = functional connection
an = plaintext
k1 = **block key**
b1 = 8-bit sequence
b2 = 7-bit index value
k2 = **cipher alphabet (128)**
cm = ciphertext

Ascertaining the ciphertext (**cm**) and retrograde searching for the plaintext (**an**) points out as an equation with two unknown veriables: **k1** und **k2.** That leads to a definite solution, only, if one unknown variable can be derived from the other unknown variable or if there are two equations with the same unknown variables.

But between the respective block key = k1 and the cipher alphabet (128) = k2 generated in the same round there are no connections. Nevertheless, even if both are extracted out of the current CypherMatrix yet they have no functional connection: (**k1** --> (**Hk MOD 169**) **+1**) and **k2** --> (**Hk +Hp**) **MOD 255**)**+1**). The current CypherMatrix itself is derivated from the initial start sequence only. But thereunto is no way back (two one-way-functions prevent this). Hence, there are many couples of cipher alphabets / block keys which result from an „brute force“ attack and deliver any legible texts, but one does not know which is the right one. Thus „brute force“ cannot have success, too.

10 Examples for testing

In appendix A some programs developed by the author are listet. For each single program the scource code may be requested from the author per e-mail (eschnoor@multi-matrix.de).

11 Notes

[#1] Algorithms, Key Sizes and Parameters Report, 3. Primitives, www.enisa.europa.eu
[#2] Singh, Simon, Geheime Botschaften. München 2004, S.298
[#3] Morin, Charles, www.kbcafe.com/articles/HowTo.Base64.pdf
[#4] Swoboda J., Spitz St., Pramateftakis M., Krytographie und IT-Sicherheit, Wiesbaden 2008, S.22
[#5] Schmeh K., Safer Net, Heidelberg1998, S.61
[#6] Paar Ch., Pelzl J., Understanding Cryptography, Berlin-Heidelberg, 2010, S.124

[#7] Strukturvergleich Klartext und Geheimtext, www.telecypher.net/Equilang.pdf
[#8] Singh S.,a.a.O., S.48:
„Coding is practised still as before acordance with principles of substitution and transposition“
[#9] Schäfer Michal B., Changeset 4524 for trunk/Templates/Experimental/CypherMatrix https://www.cryptoll.oeg/trac/CrypRool2/changeset/.../CypherMatrix.xml
[#10] Bauer F.L., Entzifferte Geheimnisse, Berlin Heidelberg NewVork 1995, S.36, 78
[#11] Schneier B., Angewandte Kryptographie (dt.Ausgabe), Bonn … 1996, S.278
[#12] Schneier B., a.a.O., S. 17
[#13] Schneier B., a.a.O., S.275
[#14] Schneier B., a.a.O., S.177
[#15] Singh S., a.a.O., Sn. 183 – 193,
[#16] Singh S., a.a.O., S. 161

Munich, in March 2014

Appendix A

According to prinziples of „Codegraphy“ the Author has developed a range of programs which are listened in the following survey:

System Basis	System Alphabet	Basis-Coding (ohne XOR-Funktion) einfache Matrix	Verbund-Coding (mit XOR-Funktion) einfache Matrix	Längen-verhältnis
1	2	Crypto01	MonoCode	1:8
2	4	Crypto02	ZweiCode	1:4
3	8	Crypto03	DreiCode	1:2,66
4	16	Crypto04	VierCode	1:2
5	32	Crypto05	QuinCode	1:1,6
6	64	Crypto06	CM64Code	1:1,33
7	128	Crypto07	DataCode DynaCryp CodeData[1)] QuadCode[2)]	1:1,143 1:1,143 1:1,143 1:1,143
8	256	Crypto08 CMCode8D System08	PlanCode MyCode08	1:1 1:1 1:1
9	512	Crypto09 System09	NeunCode MyCode09	1:1,79 1:1,79
10	1024	Crypto10 System10	ZehnCode MyCode10	1:1,6 1:1,6
11	2048	Crypt11A Crypt11B System11	ElvaCode MyCode11	1:1,46 1:1,46
12	4096	Crypto12 System12	MegaCodA MegaCodB MyCode12	1:1,33 1:1,33 1:1,33
13	8192	System 13	MyCode13	1:1,23
14	16384	System14	MyCode14	1:1,143

[1)] Programm mit drei Operationen (XOR – bit conversion - exchange),
[2)] Programm mit vier Operationen (dyn24 – XOR – bit conversion – exchange).

Each single program may with or without sourcecode per e-mail requested at the Author and tested or further developed within the scope of **CMLizenz**.

All programs are Dos-programs and will run under WindowsXP only. They have to be converted to **C#** as already be done under direction of Prof. **Bernhard Esslinger** [#1] (Universität of Singen) and his team (especially: **Michael Schäfer**) with the programs: **DataCode** and **DynaCode.** All other programs are still to be converted.

[#1] Esslinger, Bernhard, Uni Siegen, http://www.cryptool,org/de/

Appendix B

L I Z E N Z

zur Weiterentwicklung und Nutzung der Software
des „CypherMatrix“ Verfahrens

CypherMatrix®

I.

„CypherMatrix“ Verfahren ist die vom Autor *Ernst Erich Schnoor, München*, gewählte Bezeichnung einer neuen Basisfunktion der Kryptographie mit folgenden Zweckbestimmungen:

1. Durchführung von Verschlüsselungen,
2. Berechnung von Hashwerten,
3. einfache und erweiterte Signaturen und
4. weitere im Einzelnen noch zu erforschende Aufgaben.

II.

„Software zur Gestaltung des Verfahrens“ ist jeder Quellcode, gleich welcher Art, der den vorstehenden Zweckbestimmungen direkt oder indirekt dient oder zu dienen bestimmt ist, im Folgenden kurz: *„Software“* genannt.

III.

Jeder Anwender kann die Software:

1. nach eigenem Ermessen anwenden,
2. die Funktionsweise uneingeschränkt studieren und an eigene Vorstellungen anpassen,
3. Kopien der Software umsonst weiter geben,
4. sowie die Software verbessern und diese Verbesserungen zur öffentlichen Diskussion stellen.

IV.

Jeder Anwender erhält das Recht, die Marke „CypherMatrix“ für seine Tätigkeiten nach vorstehendem Abschnitt III zu benutzen.

V.

Für kommerzielle Verwendung der Software ist die vorherige schriftliche Zustimmung des Autors erforderlich. Als kommerzielle Verwendung gilt jede entgeltliche Übertragung der Software, in welchem Umfang auch immer, und der mit Hilfe der Software hergestellten Produkte, ganz oder in Teilen. Ein Verstoß gegen diese Auflage hat eine Schadensersatzzahlung des Verursachers zur Folge.

München, den 27. Mai 2011

Ernst Erich Schnoor
(eschnoor@multi-matrix.de)