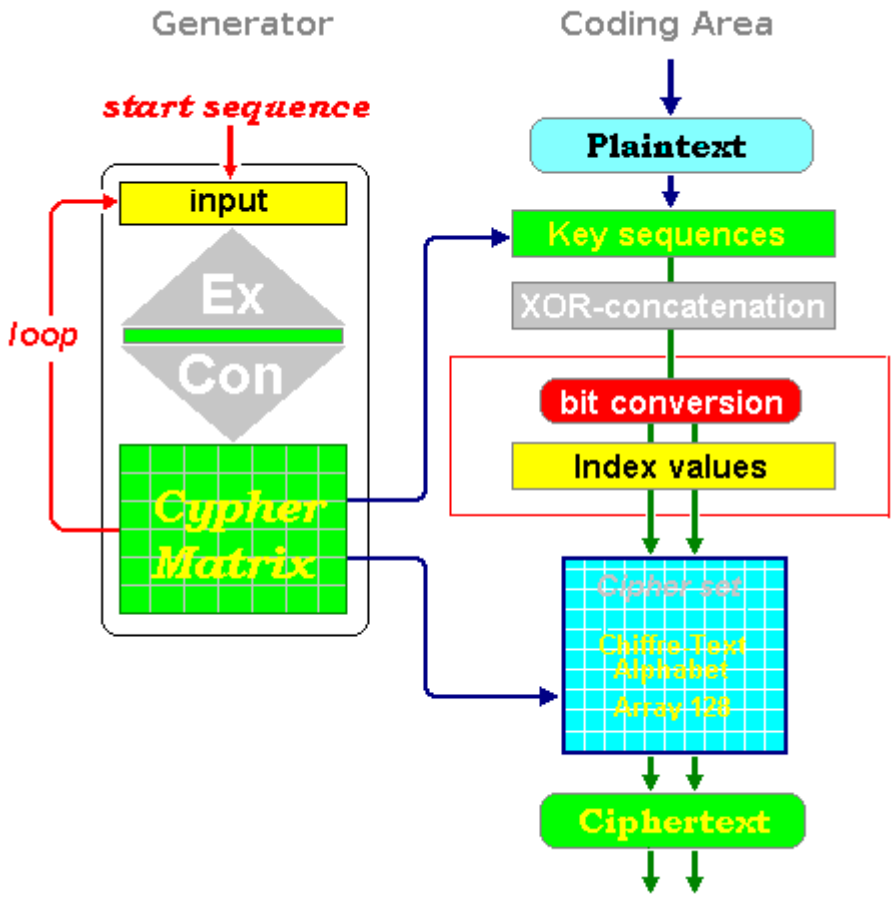
Data Encryption with >CypherMatrix<

(Ernst Erich Schnoor)

The procedure concerns dynamic and symmetric encryption of digital Data.
The **CypherMatrix** encryption [#1] is a quite new algorithm compared with actual procedures, like AES, DES, IDEA, Blowfish, RCA and others. In the broadest sense there may be seen a remote resemblance with "Coding Base 64". The encryption occurs in two independent sectors: **Generator** (Basic function) and **Coding area.** A detailed presentation of Basic function may be found in article: "Cryptographic Basic Function in Byte Techniques" .

Both sectors are combined together, but can be used separate from each other, as well.
The essential task of the **Generator** is to supply the procedure with several destination Data necessary for encryption. The actual encryption is performed in the **Coding Area.**

The following scheme demonstrates the connections:



A. >CypherMatrix< as Generator

The procedure runs dynamic because the generator creates independent destination Data for each block (e.g. 63 bytes) in any cycle. The procedure is symmetric because sender and recipient have to insert an identical start sequence (passphrase, optimal 42 characters) in order to initialize the procedure.

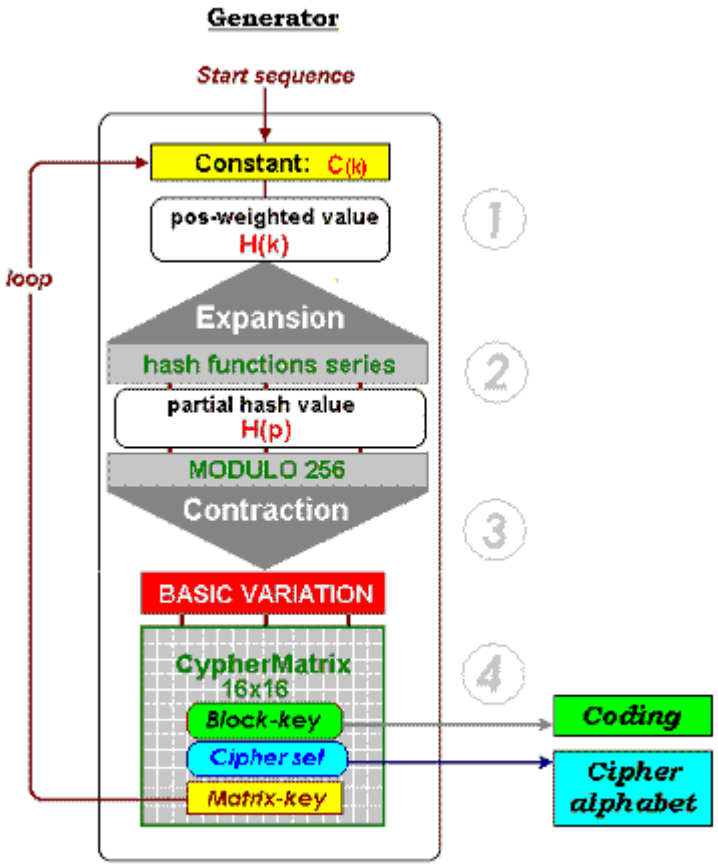
The start sequence is position weighted, multiplied with hash constant **Ck**, expanded to a hash function series (**Ex**), again contracted to BASIC VARIATION (**Con**) and finaly subjected to threefold permutation. In each cycle the destination Data and the respective **CypherMatrix** (16x16 elements) will be calculated anew. The CypherMatrix serve with the **cipher text alphabet** (128 characters), the **block key** (63 characters) to perform the XOR-concatenation and the **matrix key** (to be led back to the beginning: **loop**) to initialize the next round. A repetition of identical Data will occur first in **256!** (faculty) = **8E+506** cases due to probability laws.

Start sequence (passphrase) comprises 36 to 64 characters (optimal **42** bytes). Some examples:

9 kangaroos jumping along the Times Square	[42 bytes]
Blue flamingos flying to Northern Sutherland	[44 bytes]
Swen Hedin is sailing around the Northpole	[42 bytes]

The passphrase should be easy to remember and possibly somehow catchy and funny which one can easy keep in mind and which may not be written down anywhere. Because of its length and peculiarity lexical attacks and iterative researching will be impossible. An attacker even isn't able to analyse parts of the start sequence neither apart nor successive because the sequence could be found **in total** only, if at all.

Following overview shows the structure of the generator:



Follows the course of Data during the first round with the start sequence:

Horse racing on the banks of San Sebastian [42 bytes]

The procedure runs the following steps:

1. Position weighting and collision free

First aim is to search for a concise mapping of the start sequence definite and collisionfree. The value **H(k)** is calculated by adequate linkage of the inserted characters in order to get a starting basis.

$$H(k) = a_1 + a_2 + a_3 + \ldots a_i + \ldots a_n$$

(values for a(i) are increased by (+1), because otherwise ASCII-zero (0) would not be considered)

$$H(k) = \sum_{i=1}^{n} (a_i + 1)$$

$$H(k) = 3901$$

But the thus calculated value **H(k)** is far away to serve as a definite mapping. To achieve evident results some more features have to be added, especially **position weighting** and **collision free** device**.**

Each character **a(i)** is **position weighted** by multiplying its value with its location **p(i)** [#3]. Time is not relevant.

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

Collisions are avoided by including the **hash constant C(k)**. The constant is defined by the length (**n**) of input sequence and an individual code (1 to 99).

$$C(k) = n * (n - 2) + code \qquad code = 1$$

$$C(k) = 42*40 + 1 = 1681$$

Derivation of **„hash constant" C(k)** you will find in internet at:**"Determinats leading to Collisionfree"** [#5]

With inclusion of „hash constant" **C(k)** the interim value **H(k)** will be calculated as follows:

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

$$H_k = 6641789$$

The calculated interim value **H(k)** with a range of **10^7 = 10000000** possibilities would not yet be sufficient for a definite mapping.

2. Expansion

To obtain more exactness an **expansion** (**hash function series**) is introduced which widens the determining factors to a voluminous scale of digits without additional inputs and without loosing the quality of being collision free. The number system of expansion can be chosen between 64 to 96. Here **base 77** is fixed. The procedure expands each sequence character to decimal value (s_i) which is changed to (d_i) a digit in number system on base 77 and combined it to **hash function series (i: 1 ... m)**. Simultaneously the procedure calculates the sum of all single results (s_i) as an additional value **H(p)** in order to fix various destination Data.

$$s_i = (a_i + 1) * p_i * H_k + p_i + code + cycle$$

$$s_i \longrightarrow d_i \quad \text{(base 77)}$$

$$Hp = d_1 + d_2 + d_3 + \ldots + d_i + \ldots d_m$$

(m = number of digits in number system on base 77)

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

$$Hp = 559291769099$$

Performing expansion of chosen input sequence (42 bytes) results to the following destination Data:

	decimal	base 62
hash constant C(k):	**1681**	**R7**
position weighted value (H_k):	**6641789**	**Rrpd**
partial serial value (H_p):	**559291769099**	**9qUVGTT**
total hash value (H_p+H_k):	**559298410888**	**9qUx8J6**

Control parameters derived from destination Data show as follows:

Variante	(H_k MOD 11) +1	=	**1**	begin reconversion
Alpha	((H_k + H_p) MOD 255)+1	=	**134**	offset cipher alphabet
Beta	(H_k MOD 169)+1	=	**90**	offset block key
Gamma	((H_p + Code) MOD 196)+1	=	**37**	offset matrix key
Theta	(H_k MOD 32) +1	=	**30**	variation reconversion

By generating hash function series the sequence " **racing "** at position 7 of the input sequence results to the following calculations:

char	pi		Hk	(ai+1)*pi*Hk		Si	base 77
	(ai+1)	(ai+1)*pi			pi+code+r		
. .	. . .	. .	. . . .	. . . . .	. . . . .	. .	. . . . .	. . . . .
r	115	7	805	6641789	5346640145	9	5346640154	**1ê7Uu9**
a	98	8	784	6641789	5207162576	10	5207162586	**1æ9ã7A**
c	100	9	900	6641789	5977610100	11	5977610111	**2G3bsB**
i	106	10	1060	6641789	7040296340	12	7040296352	**2kLH2C**
n	111	11	1221	6641789	8109624369	13	8109624382	**2ërciD**
g	104	12	1248	6641789	8288952672	14	8288952686	**34zNfE**

The hash function series **H(p)** results in 248 digits in number system 77 as follows:

Dz293gO#Z4àEDD51ApWY61JRnk7bVht81ê7Uu91æ9ã7A2G3bsB2kLH2C2ërciD34zNfE14
4HtF3àJxEG46ièkH1MwdiI4âzlAJ4n7PLK4wCnâL1lrãtM57zé0N5MRE0O6KRáoP6RuJKQ
78æAKR288ZXS7WR9nT75äRuU2Q&vMV6E9xSW7YëzuX8t8nGY2pw9XZ70l0ga8wddEb8vTG
ác8ä7GgdA&à0FeBF9ëWfAV7ã8g9áCC7hBX@LLi

3. Contraction

Next step the hash function series **H(p)** will be contracted by **modulo 256** to the array **BASIC VARIATION** with 16x16 decimal digits. By this the hash function series is assumed to be a series of digits in number system on **base 78** (expansion base 77+1).

First reconversions beginning at **variante** = **1** show as follows:

Dz293gO . . . . .

3 digits base 78	decimal	MODULO 256	- theta	element
Dz2	83852	140	30	**110**
z29	371289	89	30	**59**
293	12873	73	30	**43**
93g	55032	248	30	**218**
3gO	21552	48	30	**18**
.....	.....	.....	....	.....

BASIC VARIATION generates the following structure:

BASIC VARIATION (256 elements)

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

Numbers of **BASIC VARIATION** (16x16) form the base for further processing.

4. Generating CypherMatrix (CM)

Decimal values are related directly to indexes of ASCII-character set and by this to the first **CypherMatrix** (**CM1**) with 16x16 elements. In order to increase exactness the elements of BASIC VARIATION are further subjected to a threefold **permutation** and thus form the third **CypherMatrix** (**CM3**).

Derivation first CypherMatrix (CM1) from BASIC VARIATION

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

The final CypherMatrix (CM3) is generated from the first CypherMatrix by threefold permutation (p). Permutation is performed by varying the index values in array **Matrix$(p,i,j)**. Three variations are possible:

1. Simple change of indexes: i

Pseudo code:

```
k = Alpha                                              initialising
FOR i = 1 TO 16
   FOR j = 1 TO 16
      Matrix$(1,i,j) = CHR$(Variation(k))          BASIC VARIATION
      INCR k                                          256 elements
      IF k > 256 THEN k = 1
   NEXT j
NEXT i

CM1Set$ = ""                  elements of first CypherMatrix (CM1)
CM3Set$ = ""                  elements of third CypherMatrix (CM3)

FOR s = 1 TO 3                                          three loops
    FOR i = 1 TO 16                                    (permutation)
        FOR j = 1 TO 16
           a = i - j
           IF a <= 0 THEN a = 16 + a
           SELECT CASE s
              CASE 1
                 CM1Set$ = CM1Set$ + Matrix$(1,i,j)   CypherString
                 Matrix$(2,a,j)  = Matrix$(1,i,j)
              CASE 2
```

```
                Matrix$(3,a,j)  = Matrix$(2,i,j)
             CASE 3
                CM3Set$ = CM3Set$ + Matrix$(3,i,j)  CypherString
          END SELECT
       NEXT j
    NEXT i
NEXT s
```

2. Shifting change of indexes: i,j

a = i - j

```
IF a <= 0 THEN a = 16 + a
SELECT CASE s
   CASE 1
      CM1Set$ = CM1Set$ + Matrix$(1,i,j)       CypherString
      Matrix$(2,a,j)  = Matrix$(1,i,j)
   CASE 2
      Matrix$(3,i,a)  = Matrix$(2,i,j)
   CASE 3
      CM3Set$ = CM3Set$ + Matrix$(3,i,j)       CypherString
END SELECT
```

3. Dynamic generating indexes: i,j

All indexes (16x16) are created in a separate index-array. **Index$(16,16)** (application of CypherMatrix function):

```
a = i - j
IF a <= 0 THEN a = 16 + a
SELECT CASE s
   CASE 1
      CM1Set$ = CM1Set$ + Matrix$(1,i,j)  CypherString
      Matrix$(2,a,j)  = Matrix$(1,i,j)
   CASE 2
      x = VAL(Index$(i,j))
      Matrix$(3,i,x)  = Matrix$(2,i,j)
   CASE 3
      CM3Set$ = CM3Set$ + Matrix$(3,i,j)  CypherString
END SELECT
```

With **variation 3** performed permutation of CypherMatrix shows as follow:

CypherMatrix (CM3)

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

```
209   19 52 22 E5 8D 80 30 82 84 CB DC 87 7C F1 B1 18   224
225   A1 16 68 ED 7B 9B FE A7 33 C2 50 0E 96 0A 1F 3E   240
241   47 00 C6 7A 53 F6 4E 5D C1 2C CF 5A 6F B6 B5 E7   256
```

The **structure** of CypherMatrix represents a definite mapping of the input sequence.
A repetition of identic structures due to probability law first occurs in **256!** (faculty) = **8E+506** cases.

5. Control parameters

Control parameters (**Alpha**, **Beta** and **Gamma)** pinpoint the respective offsets in the CypherMatrix from where a **partial quantity** of bytes are to be taken out necessary for processing several encryption steps.

a) Cipher text alphabet

Extracting the partial quantity „**Alphabet**" (128 characters) is arranged as from control parameter **Alpha** (ASCII-signs: 0–32,34,44,176,177,178,213,219,220,221,222 and 223 will be masked out). In our case **Alpha** = **134** takes the following bytes as cipher alphabet:

Cipher alphabet (hexadecimal)

```
FF 98 9F D1 79 AB .. CD B8 C0 F2 D7 BA D8 .. 46
CA EF .. 60 A4 59 .. 45 78 E4 9D .. 9E A9 EA 31
E1 .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
.. .. .. .. .. 8B FB FA 69 F9 .. 56 DA 38 51 ..
44 .. .. 2B 58 86 81 23 F8 55 5B B7 B9 BF 6C 2F
57 .. 7F C4 F3 26 .. D4 BE 66 .. 3B 8F 90 28 AD
.. 7D 2E 91 A6 A5 E9 3C B3 .. 6E D0 E2 63 94 9A
A8 .. .. A0 C8 54 .. 8A A2 34 83 EC 49 8E BB 4A
.. 52 .. E5 8D 80 30 82 84 CB .. 87 7C F1 .. ..
A1 .. 68 ED 7B 9B FE A7 33 C2 50 .. 96 .. .. 3E
47 .. C6 7A 53 F6 4E 5D C1 .. CF 5A 6F B6 B5 E7
```

Cipher alphabet (ASCII characters)

```
  1   ‹ û ú i ù V Ú 8 Q D + X † □ # ø    16
 17   U [ · ¹ ¿ l / W □ Ä ó & Ô ¾ f ;    32
 33   □ □ (   } . ` ¦ ¥ é < ³ n Đ â c    48
 49   ″ š ¨   È T Š ¢ 4 ƒ ì I Ž » J R    64
 65   å □ € 0 , „ Ë ‡ | ñ ¡ h í { › þ    80
 81   § 3 Â P – > G Æ z S ö N ] Á Ï Z    96
 97   o ¶ µ ç □ ˜ Ÿ Ñ y « Í ¸ À ò × °   112
113   Ø F Ê ï ` ¤ Y E x ä □ ž © ê 1 á   128
```

b) Block key

To perform XOR-concatenation with **plain text blocks** (7 to 70 bytes: in present case **63** characters) the control parameter **Beta** fixes the point from where byte series in block length (63 bytes) are taken from the CypherMatrix as current **block key.**

In the first round the program generates the following block key:

Block key (from offset: Beta = 90)

```
.. .. .. .. .. .. .. .. .. 37 73 6A BD 39 9C AC 95 88 06 21 67 97
DE F0 25 5F B2 EB AA 5E 85 0F 40 B0 20 C7 89 D2 13 99 09 B4 EE 5C
12 64 D9 E6 3D 93 43 1B 3A 8B FB FA 69 F9 02 56 DA 38 51 DB 44 14
1D 2B 58 86 81 23 .. .. .. .. .. .. .. ..
 7sj½9œ¬•ˆ#!g—Þð%_²ëª^…#@° Ç‰Ò#™´î\#dÙæ="C#:‹ûúiù#VÚ8QÛD##+X†□#
```

Length of plain text block and by this length of the the block key, as well can be chosen for each programm between 7 up to 70 bytes (multible of 7) – here chosen **63**) . Block keys are necessary in encryptions with XOR-concatenations, only.

c) Matrix key

From parameter **Gamma** the program takes the **start sequence** with **42** bytes to initialize the next round.

Matrix key (from offset: Gamma = 37)

```
.. .. .. .. FD D3 4D F7 C5 77 7E 65 AF E3 92 17
32 75 41 CE 62 4B 36 AE E8 FC C9 A3 08 8C F5 48
E0 3F 27 61 76 2D 04 35 10 2A F4 BC 72 6B .. ..

ýÓM÷Åw~e¯ã'#2uAÎbK6®èüÉ£#ŒõHà?'av-#5#*ô¼rk
```

Matrix key is led back to the beginning of the program (**loop**) and serves for initializing the next cycle. The series of round matrix keys control the total course of the procedure identically at sender and recipient.

6. Further rounds

By means of matrix key from the foregoing cycle (**loop**: second and further rounds) the program always generates a new **CypherMatrix** and new **Control parameters**. In present case for example:

	Variante	Alpha	Beta	Gamma	Theta	Cipher alphabet
2. round	10	157	155	87	2	8D DA F2 E1 F1 …..
3. round	3	157	75	49	18	E4 50 8A 7D 2D …..
4. round	5	110	108	71	31	79 69 BE D9 8D …..
. . . . .	…	…	…	…	…	……

The **plaintext** to be encrypted and the resulting **cipher text** are of no influence on the control parameters to be generated. Both areas – generator and coding area – are connected by „**block key**“ and „**cipher alphabet**“ only.

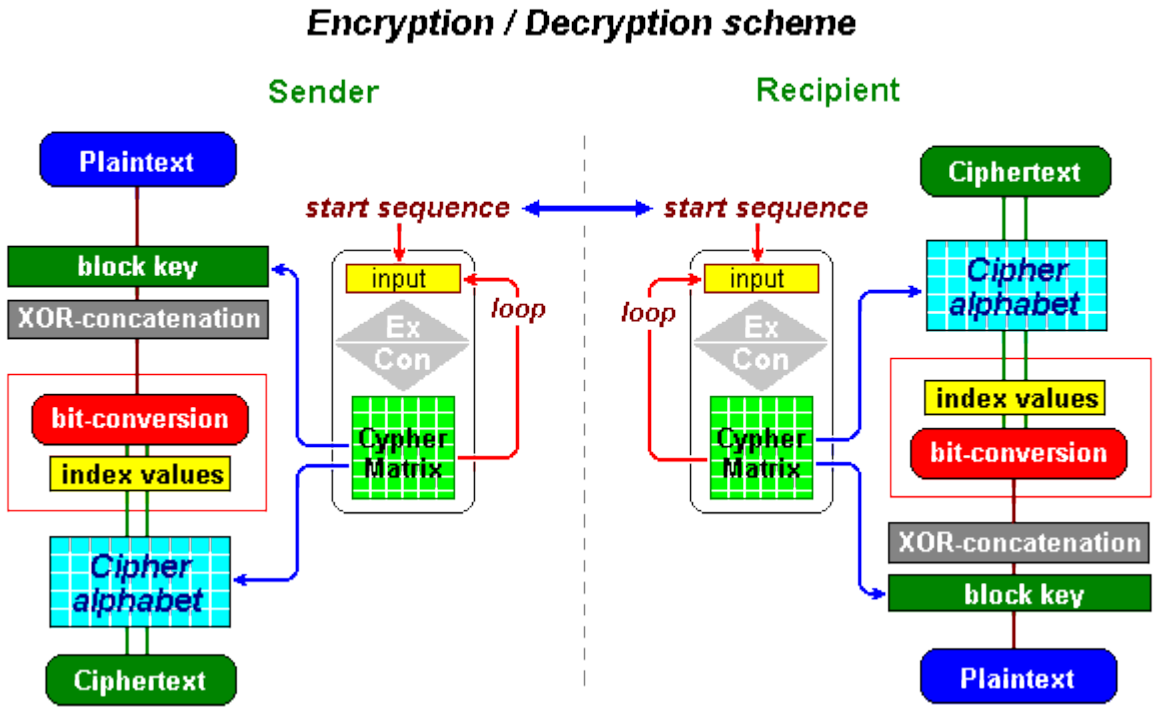
B. Encrypting in Coding Area

Encrypting at sender and recipient as well is processed in all rounds completely by control parameters **Alpha, Beta** and **Gamma** supplied from the generator.

Encryption is performed in three internal functions:

1. Partial dynamic „one-time-pad“
   plain text block → block key → 8-bit XOR concatenation
2. Bit conversion
   8-bit XOR concatenation → 7 bit index values (0 ...127)
3. Destination of Cipher text
   7-bit index values → Cipher alphabet (0...127) → Cipher text.

The following scheme shows the connections:



In each round plaintext block (here chosen with 63 bytes) and block key with identical length extracted out of the CypherMatrix are **XOR** concatenated. This results in principle to a partial „**one time pad**“. The key will not be repeated because a different key is created in each round.
Following course of encryption will be demonstrated by way of text file CANNERY.TXT (401 bytes)

> *The WORD is a symbol and a delight which sucks up men and scenes, trees, plants, factories, and Pekinese. Then the Thing becomes the Word and back to Thing again, but warped and woven into a fantastic pattern. The word sucks up Cannery Row, digests it and spews it out, and the Row has taken the shimmer of the green world and the sky-reflecting seas.*
>
> *John Steinbeck, Cannery Row, New York 1945*

By inserting the start sequence "**Horse racing on the banks of San Sebastian**" the control parameters - generated in section A - result to following course in the first round:

1. XOR concatenation

XOR concatenation as first step runs as follows:

Plain text (63 bytes)

The WORD is a symbol and a delight which sucks up men and scene

```
54 68 65 20 57 4F 52 44 20 69 73 20 61 20 73 79 6D 62 6F 6C 20
61 6E 64 20 61 20 64 65 6C 69 67 68 74 20 77 68 69 63 68 20 73
75 63 6B 73 20 75 70 20 6D 65 6E 20 61 6E 64 20 73 63 65 6E 65
```

Block key (63 Bytes)

```
37 73 6A BD 39 9C AC 95 88 06 21 67 97 DE F0 25 5F B2 EB AA 5E
85 0F 40 B0 20 C7 89 D2 13 99 09 B4 EE 5C 12 64 D9 E6 3D 93 43
1B 3A 8B FB FA 69 F9 02 56 DA 38 51 DB 44 14 1D 2B 58 86 81 23
```

XOR concatenation(63 Bytes)

```
63 1B 0F 9D 6E D3 FE D1 A8 6F 52 47 F6 FE 83 5C 32 D0 84 C6 7E
E4 61 24 90 41 E7 ED B7 7F F0 6E DC 9A 7C 65 0C B0 85 55 B3 30
6E 59 E0 88 DA 1C 89 22 3B BF 56 71 BA 2A 70 3D 58 3B E3 EF 46
```

2. Bit conversion

The second step, **bit conversion**, is data techical a change of bits. In an analytically point of view it is a change of bytes in bytesystem on base 8 (**8 bit**) to bytes in bytesystem on base 7 (**7 bit**). It is a step which regroupes the results of XOR concatenation from 8-bit segments into 7-bit segments (values from 0 to 127). Numbers and series of digits „0" and „1" remain unchanged. No bit is added and no bit is omitted. A new grouping (8-bit → 7-bit) is arranged, only.

Bit-Konversion

Changing 8-bit XOR sequences to 7-bit sequences (Indexes:0-127)

8-bit:

```
01100011 00011011 00001111 10011101 01101110 11010011 11111110 11010001
10101000 01101111 01010010 01000111 11110110 11111110 10000011 01011100
00110010 11010000 10000100 11000110 01111110 11100100 01100001 00100100
```

7-bit:

```
0110001 1000110 1100001 1111001 1101011 0111011 0100111 1111110 1101000
1101010 0001101 1110101 0010010 0011111 1101101 1111110 1000001 1010111
0000110 0101101 0000100 0010011 0001100 1111110 1110010 0011000 0100100
100....
```

Hexadecimal 7-bit sequences as index values (0-127)

```
31 46 61 79 6B 3B 27 7E 68 6A 0D 75 12 1F 6D 7E 41 57 06 2D 04 13 0C
7E 72 .. .. .. ..
```

By next step the decimal values of 7-bit sequences become index values for positions of cipher characters in the cipher alphabet. The index values have to be increased by **+1** because the alphabet array accepts index values **1 – 128**, only.

Decimal index values (+1) for Ciphertext alphabet

```
50 71 98 122 108 60 40 127 105 107 14 118 19 32 110 127
66 88 7 46 5 20 13 127 115 ...
```

Encrypted ciphertext

```
9A CB B6 E4 B8 49 A6 31 79 CD 81 A4 B7 3B F2 31 8D C6 DA D0 F9 B9 86 31 CA
7F 7D F1 FA 38 FE F2 4E 5A 31 DA 59 CA C8 A9 A8 30 2F 51 3C 47 9F 94 A2 2F
8E 51 CB 79 83 44 5B 23 45 A4 A0 CB 60 3C 34 F8 B3 69 5A F8 CF CB 0D 0A 7F
FE F4 7A E0 2D 65 9A 33 2F EE 47 C5 46 64 5D 98 74 6C FE 65 FA B7 86 49 CF
93 54 D9 30 94 46 51 4B FE 36 8C 2D 75 68 40 CC 82 5D 5F EE 47 CF E9 98 62
49 4F 5A 2F 51 33 8C 79 8C 8E 74 3D BD 49 49 D0 AC 6F E9 4E A9 0D 0A 47 EC
30 C0 69 96 51 A7 4E DA 54 28 24 41 ED BA A0 4C C0 EA 49 28 3F 81 ED 6E CF
A1 FA 5C 24 4E A1 A9 F4 5C 32 84 9B AA 5E D0 FD D4 25 67 23 FE FE E4 41 EA
6E C8 52 AF 96 31 31 52 6B E2 24 E4 B6 56 30 85 51 31 D1 B8 0D 0A B3 F2 42
77 4A 23 E2 D8 43 E1 C6 4D 21 E2 23 A2 92 FC E4 F2 42 67 BA E2 62 E3 73 3D
```

```
šË¶ä¸I¦1yÍ¤·;ò1ÆÚÐù¹†1Ê}ñú8þòNZ1ÚYÊÈ©¨0/Q<GŸ”¢/ŽQËyƒD[#E¤ Ë`<4ø
³iZøÏË þôzà-eš3/îGÅFd]˜tlþeú·†IÏ“TÙ0”FQKþ6Œ-
uh@Ì,]_îGÏé˜bIOZ/Q3ŒyŒŽt=½IIÐ¬oéN©Gì0Ài-Q§NÚT($Aí° LàêI(?ínÏ¡ú\
$N¡©ô\2„›ª^ÐýÔ%g#þþäAênÈR¯-11Rkâ$ä¶V0…Q1Ñ¸³òBwJ#âØCáÆM!â#
¢’üäòBg°âbãs=yƒðy ...
```

As a result of „**bit conversion**“ there is a fundamental effect in the procedure:
Length of cipher text related to length of plain text prolongs in ratio 7:8. Primary 7 plain text characters become 8 cipher text characters. By this the demand plain text and cipher text should have the same length cannot be observed. Each plain text character fall upon 8/7 cipher text characters. Nonetheless there are programs developed in CypherMatrix algorithm which lead to equal length between plain text and cipher text: Storing Space Convergence

C. Decryption

Decryption of cipher texts goes the reverse way than encryption. Generating of control parameters are achieved as demonstrated in section **A**. By this an identical performace arises as at encryption, but in reverse order, only.

1. Analysing Cipher text
   Cipher text → Cipher text alphabet (0...127) → 7 bit index values
2. Bit conversion
   7 bit index values (0 ...127) → 8-bit XOR concatenation
3. XOR concatenation
   8-bit XOR concatenation → block key → plain text block

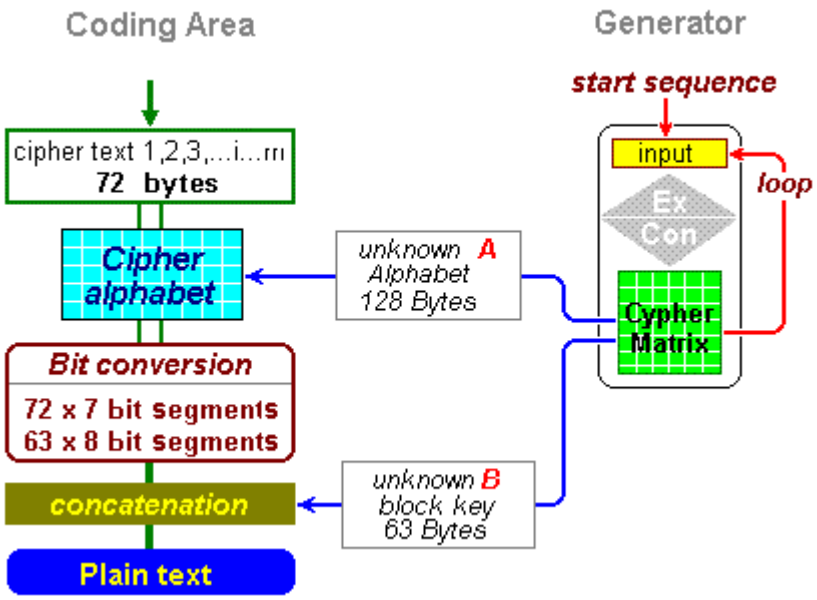
For blocks of **72** bytes cipher text the program takes the index values of single characters from the respective cipher alphabet and combines these 7-bit sequences to a series of **504**

bits which are in return devided into **63** 8-bit sequences. The resulting string is XOR concatenated with the respective **block key,** thus the original plain text turns up again.

D. Security of the Procedure

Security of the encryption process may be judged by the following facts:
Today the attacks on encrypted information presuppose almost all that plain text and cipher text have identical length. In this case only usual attacks, like structure analysis, known plaintext attack, chosen plaintext attack, differential and linear cryptanalysis etc. may be successful. But because CypherMatrix encryption has no „**congruence of length**“ all this attacks will be beyond all hope. „**Bit conversion**“ breaks through the functional connection between cipher text and plain text. Even „brute force“ will have no chance.

Principally an attacker knows the **Ciphertext** and the respective **length** of the encrypted message, only. Further he may know the "CypherMatrix" method with its three functions, but he does'nt know the actual control parameters: **Alpha**, **Beta** and **Gamma**. Because he does not know even the **start sequence** he will be unable to break the cipher. He only may try to iterate all possibilities. An attack on the start sequence of **42 bytes** yields in an exponential complexity of **O(2^336)** = **1.4E+101**. „Brute force“ attack will be hopeless.



The parameters **block key** (**B**) and **ciphertext array** (**A**) are two variables independent from each other.

Cipher text = **f [** f1 (plain text,**k1**), f2 (b1,b2), f3 (b2,**k2**) **]**
Plain text = **f [** f3 (cipher text,**k2**), f2 (b2,b1), f1 (b1,**k1**) **]**

fx = function
k1 = block key
b1 = 8-bit sequence
b2 = 7-bit index-value
k2 = cipher text array (128)

Both, encryption and decryption form equations with two unknown variables: **k1** and **k2**. This results in a definite solution only if one of the unknown can be derived from the other unknown variable or if there are two equations with the same unknown variables. But there is no connection between the respective block key= **k1** and the array (cipher alphabet 128)= **k2** generated in the same cycle. Both are extracted out of the current CypherMatrix but they have no functional connection (**k1** → (**Hk MOD 169**)+1) and (**k2** → (**Hk+Hp MOD 255**) +1). Both their common roots are the initial **start sequence** only. But to that position is no way back (two one way functions prevent this).

D. Combined Operationen

An encryption program constructed by the foregoing demonstrations is one of the simplier solution. XOR concatenation and bit conversion may be combined or replaced with further operations (e.g. „dyn24“, „bit exchange“, „bit crossing“, byte-tansposition etc.). There are no borders for variety. Detailed information are in article: "Core of the CypherMatrix procedure"

Munich, in June 2010

[#1] Autors patent: DPMA 19811593 as of 18.03.1998
[#2] http://www.telecypher.net/Bytetechniques.pdf
[#3] analogous to: Descartes, René, (without more indication)
[#5] Collisionfree: telecypher.net/Collfree.pdf