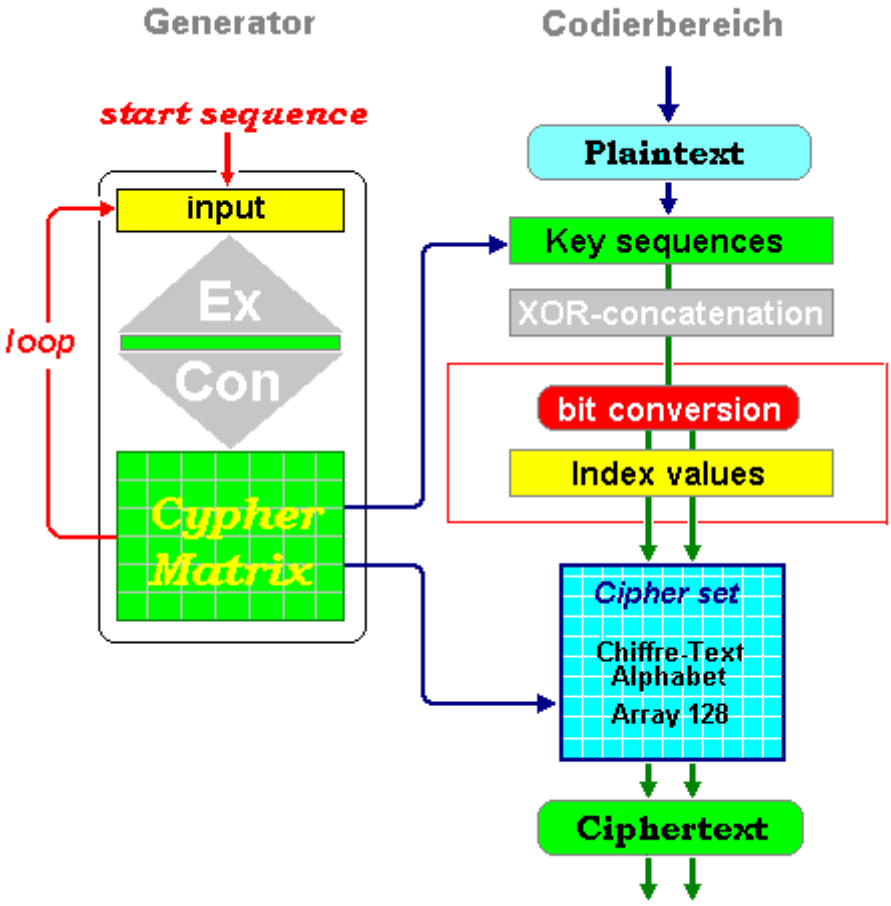
Verschlüsseln mit >CypherMatrix<

(Ernst Erich Schnoor)

Es handelt sich um eine dynamische symmetrische Verschlüsslung digitaler Daten.
Verglichen mit aktuellen Verfahren – wie AES, DES, IDEA, Blowfish, RC4 und andere - ist die **Cypher Matrix** Verschlüsselung [#1] ein völlig neues Verfahren. Im gewissen Sinne kann noch ein Vergleich mit "Coding Base 64" gesehen werden. Die Verschlüsselung geschieht in zwei unabhängigen Bereichen: **Generator** (mit Basisfunktion) und **Codierbereich**. Eine ausführliche Darstellung der Basisfunktion findet sich im Artikel: Kryptographische Basisfunktion in Byte-Technik.

Beide Bereiche werden kombiniert, können aber auch getrennt verwendet werden. Aufgabe des **Generators** ist die Bereitstellung von Steuerungsparameter für den Ablauf des Verfahrens. Die eigentliche **Verschlüsselung** vollzieht sich im Codierbereich. Das folgende Schema erläutert die Zusammenhänge:



A. >CypherMatrix< als Generator

Das Verfahren ist dynamisch, weil der Generator für jeden Block (z.B. 63 Bytes) in jeder Runde neue unabhängige Steuerungsparameter generiert. Das Verfahren ist symmetrisch, weil Sender und Empfänger zur Initialisierung des Generators die gleiche Startsequenz eingeben (Passphrase, optimal 42 Zeichen). Die Eingangssequenz wird positionsgewichtet,

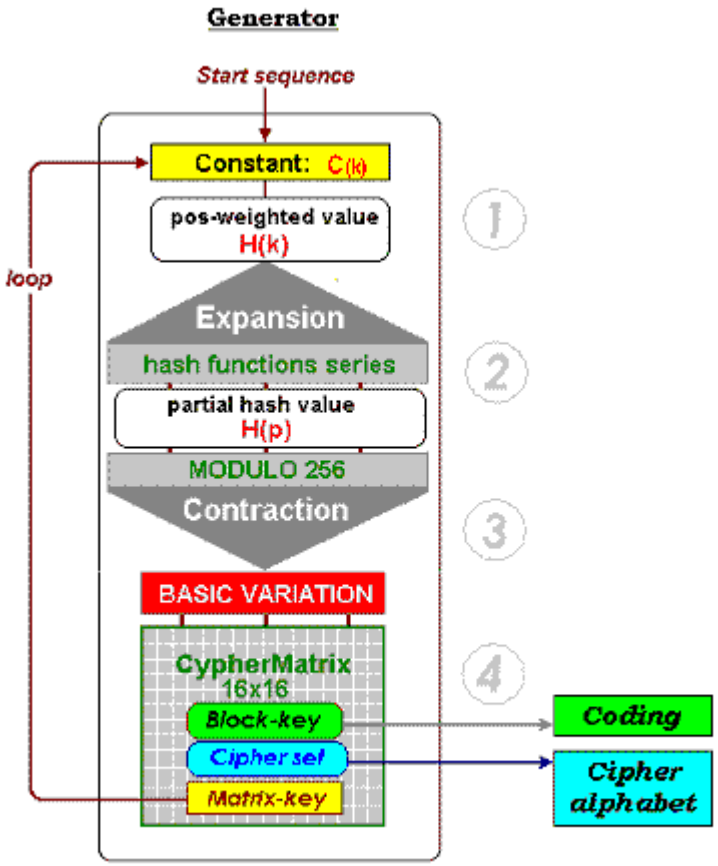
mit der Hashkonstante **Ck** multipliziert, zu einer Hashfunktionsfolge erweitert **(Ex)**, wieder zur BASIS VARIATION verdichtet (**Con**) und einer dreifachen Permutation unterworfen. In jeder Runde werden die Steuerungsparameter neu errechnet und die jeweilige Runden **CyperMatrix** (16x16 Elemente) neu generiert. Sie liefert das **Chiffretext Alphabet** (128 Zeichen)**,** den **Block-Key** (63 Zeichen) zur XOR-Verschlüsselung und den **Matrix-Key** (zurückgeführt auf den Beginn: **loop**) zur Initialisierung der nächsten Runde. Nach den Gesetzen der Wahrscheinlichkeit entsteht die Wiederholung einer gleichen **CypherMatrix** zur Generierung der Steuerungsparameter erst in **256!**(Fakultät) = **8E+506** Fällen.

Die Startsequenz (Passphrase) umfasst 36 bis 64 Zeichen (optimal **42** Bytes). Einige Beispiele:

Die Schildbürger fegen den Teutoburgerwald	[42 Bytes]
Bruno der Braunbär aus Bregenz im Breisgau	[42 Bytes]
9 kangaroos jumping along the Times Square	[42 Bytes]
Blue flamingos flying to Northern Sutherland	[44 Bytes]

Die Passphrase sollte ungewöhlich und dennoch leicht zu behalten sein, so dass sie nicht aufgeschrieben werden muss aber auch nicht geraten werden kann. Wegen ihrer Länge und Eigenart kann sie weder durch Iteration noch durch einen Wörterbuchangriff analysiert werden. Ein Angreifer kann auch nicht mit Erfolg versuchen, Teile des Schlüssels getrennt oder nacheinander zu brechen, da die Startsequenz nur in einem **Durchgang als Ganzes** gefunden werden kann, wenn überhaupt.

Die folgende Übersicht zeigt die Struktur des Generators:



Im Folgenden wird der Datenverlauf in der ersten Runde mit der folgenden Startsequenz gezeigt:

Die Schildbürger fegen den Teutoburgerwald [42 Bytes]

Die Eingabesequenz durchläuft die folgenden Stufen:

1. Positionsgewichtung

Gesucht wird eine prägnante Abbildung der Eingabesequenz eindeutig und kollisionsfrei. Als Ausgangsgrundlage wird durch geeignete Verknüpfung der eingegebenen Zeichen zunächst der Wert **H(k)** berechnet:

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

(der Wert für a(i) erhöht sich um (+1), da sonst ASCII-null nicht berücksichtigt wird)

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

$$\mathbf{H(k) = 4130}$$

Der für **H(k)** errechnete Wert ist jedoch weit davon entfernt, als geeignete Grundlage zu dienen . Es müssen weitere Merkmale hinzu kommen, insbesondere **Positionsgewichtung** und **Kollisionsfreiheit,** um eindeutige Ergebnisse zu erzielen.

Jedes Zeichen der Sequenz **a(i)** wird **positionsgewichtet,** indem sein Wert mit seiner Position **p(i)** multipliziert wird. Die Zeit ist nicht relevant und wird mit **t = 1** gesetzt.

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

Kollisionen werden durch Einbindung der **Hashkonstante C(k)** verhindert.
Die Konstante **C(k)** bestimmt sich allein aus der Länge (**n**) der Eingabesequenz und einem individuellen Anwender Code (1 bis 99).

$$\mathbf{C(k) = n * (n - 2) + code \qquad code = 1}$$
$$\mathbf{C(k) = 42*40 + 1 = 1681}$$

Die Ableitung der Hashkonstanten **C(k)** wird im Artikel "Bestimmungsfaktoren für Kollisionsfreiheit" dargelegt. Unter Einbindung der „Hashkonstanten“ **C(k)** wird der Zwischenwert **H(k)** wie folgt errechnet:

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

$$\mathbf{H_k = 7033932}$$

Der errechnete Zwischenwert **H(k)** mit einer Spanne von **10^7 = 10000000** Möglichkeiten ist für eine eindeutige Abbildung noch unzureichend.

2. Expansion

Zur Erhöhung der Sicherheit wird die **Hashfunktionsfolge H(p)** eingeführt, die die Eingangssequenz zu einer umfangreichen Folge in einem höherwertigen Zahlensystem expandiert. Das Zahlensystem der Expansion ist wählbar zwischen 64 bis 96. Hier wird **Basis 77** festgelegt. Für jedes Zeichen der Eingabe-Seqzenz errechnet das Verfahren den dezimalen Wert (s_i), der dann zu (d_i) (Ziffern im Zahlensystem zur Basis 77) umgewandelt wird. Gleichzeitig ermittelt das Verfahren die Summe aller einzelnen Ergebnisse (s_i) als zusätzlichen Abbildungswert **H(p)** zur Bestimmung verschiedener Steuerungsparameter.

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

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

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

(m = Anzahl der Zahlen im System zur Basis 77)

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

$$Hp = 642915453651$$

Als Bestimmungsdaten ergeben sich die folgenden Werte:

	dezimal	Basis 62
Hashkonstante C(k):	**1681**	**R7**
positionsgewichteter Wert (H_k):	**7033932**	**TVqW**
partieller Zwischenwert (H_p):	**642915453651**	**BJlnit9**
gesamter Hashwert (H_p+H_k):	**642922487583**	**BJmHEjf**

Aus den Bestimmungsdaten errechnet das Verfahren die folgenden Steuerungsparameter:

Variante	(H_k MOD 11) +1	=	**5**	Beginn Rückumwandlung
Alpha	((H_k + H_p) MOD 255)+1	=	**49**	Offset Chiffre-Alphabet
Beta	(H_k MOD 169)+1	=	**153**	Offset Block-Schlüssel
Gamma	((H_p + Code) MOD 196)+1	=	**193**	Offset Matrix-Schlüssel
Delta	((H_k + H_p) MOD 155)+1	=	**104**	dynamische Bit-Folgen
Theta	(H_k MOD 32) +1	=	**13**	Variation Rückumwandlung

Bei der Generierung der Hashfunktionsfolge ergeben sich beispielsweise für die Zeichen in den Positionen **18** bis **22** der Startsequenz (...fegen...) die folgende Berechnung:

Zchn	pi		Hk	(ai+1)*pi*Hk		Si	Basis 77
	(ai+1)	(ai+1)*pi			pi+code+r		
. .	. . .	. .	. . . .	. . . . .	. . . . .	. .	. . . . .	. . . . .
f	103	18	1854	7033932	13040909928	20	13040909948	4&ê7gá
e	102	19	1938	7033932	13631760216	21	13631760237	52yNàl
g	104	20	2080	7033932	14630578560	22	14630578582	5VFB44
e	102	21	2142	7033932	15066682344	23	15066682367	5hkUVi
n	111	22	2442	7033932	17176861944	24	17176861968	6QmijZ
..	. . .	. .	. . . .	. . . . .	. . . . .	. .	. . . . .	. . . . .
					Summe:		642915453651	36e3Jx9

Die **Hashfunktionsfolge H(p)** umfasst **248** Ziffern im Zahlensystem zur Basis 77:

D&7âågWQYczHmQfQVwkH1734Yå1h4SRà1å5R3N2FqSyä2gMfv#2m7RG42#äerk44BRfd3ã
Aã7t3yQ02à3êBAth4yDUd81ZJbN84&ê7gá52yNàl5VFB445hkUVi6QmijZ1éâ9rä6N2J7r
6mIhfJ7caVrt2OLãN76EHPI27qâuoæ9FPxeã9WvFäh9OArUm8bsj6vAWy0ukAZTNWi9uBá
7L9&C5åLBRV&#éCCY5ldAESfhEBlHAeáB1zqpç

3. Kontraktion

Als nächste Stufe wird die Hashfunktionsfolge H(p) mit **Modulo 256** zum Array **BASIS VARIATION** in 16x16 dezimale Ziffern verdichtet (ohne Wiederholung). Dabei wird für die Hashfunktionsfolge das Zahlensystem zur Basis **78** unterstellt (Expansion 77 + 1).

Die ersten vier Rückrechnungen ab **Variante** = **5** zeigen sich wie folgt:

. . . . **ågWQYc . . . .**

3 Ziffern Basis 78	dezimal	Modulo 256	- Theta	Element
. . . .	. . . . .	. . .	. . .	. . .
ågW	429188	132	13	**119**
gWQ	258050	2	13	**245**
WQY	196750	142	13	**129**
QYc	160874	106	13	**93**
. . . .	. . . . .	. . . .	. . .	. . . .

Durch die Verdichtung entsteht das Array **BASIS VARIATION** mit 256 Elementen:

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

Die Zahlen der **BASIS VARIATION** (16x16) bilden die Grundlage für die weiteren Schritte.

4. Generierung der CypherMatrix

Die dezimalen Zahlen der BASIS VARIATION (16x16) können direkt auf die 256 Werte des erweiterten ASCII-Zeichensatzes bezogen werden und ergeben somit die erste **CypherMatrix** mit 16x16 Elementen.

CypherMatrix (CM1)

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

Aus der ersten CypherMatrix (CM1) wird durch dreifache Permutation (**p)** die endgültige CypherMatrix (CM3) gebildet. Die Permutation entsteht durch Variation der Indexwerte im Array **Matrix$ (p,i,j).** Dabei sind drei Varianten möglich:

1. **Einfacher Austausch Index: i**
   Pseudo-Code:

```
CM1Set$ = ""                 Elemente der ersten CypherMatrix (CM1)
CM3Set$ = ""                Elemente der dritten CypherMatrix (CM3)

FOR s = 1 TO 3                                          drei Schleifen
    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. Versetzter Austausch Indizes: 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. Dynamische Generierung Indizes: i,j

Alle Indexwerte (16x16) werden in einem gesonderten Index-Array **Index$(16,16)** neu generiert (Anwendung der CypherMatrix Funktion):

```
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
```

Die mit **Variante 3** permutierte CypherMatrix zeigt sich wie folgt:

CypherMatrix (CM3)

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

Die **Struktur** der CypherMatrix stellt eine eindeutige Abbildung der Eingabesequenz dar. Nach den Gesetzen der Wahrscheinlichkeit tritt eine Wiederholung der gleichen Struktur

erst in **256!** (Fakultät) = **8E+506** Fällen ein. Aus der CypherMatrix (CM3) werden alle für die Verschlüsselung erforderlichen Steuerungsparameter entnommen.

5. Steuerungsparameter

In jeder Runde bewirken die Steuerungsparameter (**Alpha**, **Beta** und **Gamma)** ab verschiedenen Offsets der CypherMatrix die Entnahme der für die Verfahrensschritte zur Verschlüsselung erforderlichen **Teilmengen** von Zeichen.

a) Chiffretext-Alphabet

Die Entnahme der Teilmenge **„Alphabet“** (128 Zeichen) geschieht ab dem Steuerungsfaktor **Alpha** (ASCII-Zeichen: 0–32,34,44,176,177,178,213,219,220,221,222 und 223 werden ausgeblendet). Im vorliegenden Fall mit **Alpha** = **49** entnimmt das Programm folgendes Chiffre-Alphabet:

Geheim-Zeichen-Alphabet (hexadezimal)

```
C2 .. 56 C3 34 B4 B3 .. 55 AF 73 6E E5 B8 F4 AC
93 .. 2F C8 .. 82 24 8B 42 45 85 78 6B 96 7E C1
EC 9B FC 36 .. 66 F8 23 .. 46 AD 37 C0 2A 69 A6
D1 30 6A CE 51 .. E9 .. B5 .. 33 67 .. 5E 4D CD
65 3B 8F E0 A3 .. EA AA BF D0 8D EF 4F 86 .. C7
28 57 .. 95 75 BB .. CC EB E7 .. AE .. .. 48 ..
61 .. F0 72 9C .. E4 5D 74 2D .. .. A9 50 CF C9
D2 80 81 .. 5F A1 A5 84 7C FE 38 F3 C5 88 BE BD
BC F5 F2 35 6F 49 64 E2 E3 B6 .. 40 F6 B9 63 99
D6 47 3D .. 59 9E 77 .. .. 25 41 .. .. .. .. ..
```

Geheim-Zeichen-Alphabet (ASCII-Zeichen)

```
  1   Â V Ã 4 ´ ³ U ¯ s n å ¸ ô ¬ " /   16
 17   È , $ ‹ B E … x k – ~ Á ì › ü 6   32
 33   f ø # F   7 À * i ¦ Ñ 0 j Î Q é   48
 49   µ 3 g ^ M Í e ; □ à £ ê ª ¿ Ð □   64
 65   ï O † Ç ( W • u » Ì ë ç ® H a ð   80
 81   r œ ä ] t - © P Ï É Ò € □ _ ¡ ¥   96
 97   „ | þ 8 ó Å ˆ ¾ ½ ¼ õ ò 5 o I d  112
113   â ã ¶ @ ö ¹ c ™ Ö G = Y ž w % A  128
```

b) Block-Schlüssel

Für XOR-Verknüpfungen mit **Klartext-Blocks** von 7 bis 70 Bytes (im vorliegenden Fall sind 63 Zeichen festgelegt) bestimmt die Variable **Beta** die Position, ab der eine Zeichenfolge in der Blocklänge (63 Zeichen) als **Block-Schlüssel** aus der CypherMatrix entnommen wird.

In der ersten Runde generiert das Programm den folgenden Block-Schlüssel:

Block-Schlüssel (ab Offset: Beta = 153)

```
.. .. .. .. .. .. .. .. 74 2D 17 12 A9 50 CF C9
D2 80 81 00 5F A1 A5 84 7C FE 38 F3 C5 88 BE BD
BC F5 F2 35 6F 49 64 E2 E3 B6 B0 40 F6 B9 63 99
D6 47 3D DF 59 9E 77 14 16 25 41 26 A4 76 A8 31
B7 6D 94 09 79 4C DC .. .. .. .. .. .. .. .. ..
```

Block-Schlüssel in ASCII-Zeichen

```
t-##©PÏÉÒ€□#_¡¥„|þ8óÅ^¾½¼õò5oIdâã¶°@ö¹c™ÖG=ßYžw##%A&¤v¨1·m” yLÜ
```

Die Länge der Klartext-Blöcke, und damit auch der Block-Schlüssel, kann für jedes Programm manuell mit 7 bis 70 Bytes (Vielfaches von 7) – hier **63** Bytes – gewählt werden. Block-Schlüssel sind nur in Verschlüsselungen mit XOR-Verknüpfung erforderlich.

c) Matrix-Schlüssel

Ab Parameter **Gamma** entnimmt das Verfahren die **Eingabesequenz** mit **42** Bytes für die nächste Runde.

Matrix-Schlüssel (ab Offset: Gamma = 193)

```
D6 47 3D DF 59 9E 77 14 16 25 41 26 A4 76 A8 31
B7 6D 94 09 79 4C DC 5C DA A7 FA 3A D9 21 32 8E
BA 9F F9 5A 20 E8 43 A0 F7 7B .. .. .. .. .. ..
```

Matrix-Schlüssel in ASCII-Zeichen

```
ÖG=ßYžw##%A&¤v¨1·m” yLÜ\Ú§ú:Ù!2Ž°ŸùZ èC ÷{
```

Der Matrix-Schlüssel wird auf den Eingang des Verfahrens zurückgeführt (**loop**) und dient zur Initialisierung des nächsten Durchlaufs. Die Folge der Runden-Matrix Schlüssel steuert den gesamten Ablauf der Verschlüsselung identisch beim Sender und beim Empfänger.

6. Die weiteren Runden

Mit dem **Matrix-Schlüssel** aus der vorhergehenden Runde (**loop:** zweite und weitere Runden) generiert das Verfahren eine neue **CypherMatrix** und neue **Steuerungs-parameter**. Im vorliegenden Fall beispielsweise:

	Variante	Alpha	Beta	Gamma	Theta	Chiffre-Alphabet
2. Runde	10	132	75	191	3	E2 3C 6E 44 2E . . .
3. Runde	4	92	133	177	1	3C 8F C6 60 ED . . .
4. Runde	4	124	91	110	16	C7 71 FD D7 FB . . .
5. Runde	9	175	50	21	25	25 9D E6 BB E9 . . .
. . . . .	…	…	…	…	…	.....

Der zu verschlüsselnde **Klartext** und der erzeugte **Chiffretext** haben keinen Einfluss auf die generierten Steuerungsparameter. Beide Bereiche – Generator und Codierbereich – sind nur durch den „**Block-Schlüssel**“ und das „**Chiffretext-Alphabet**“ verbunden.

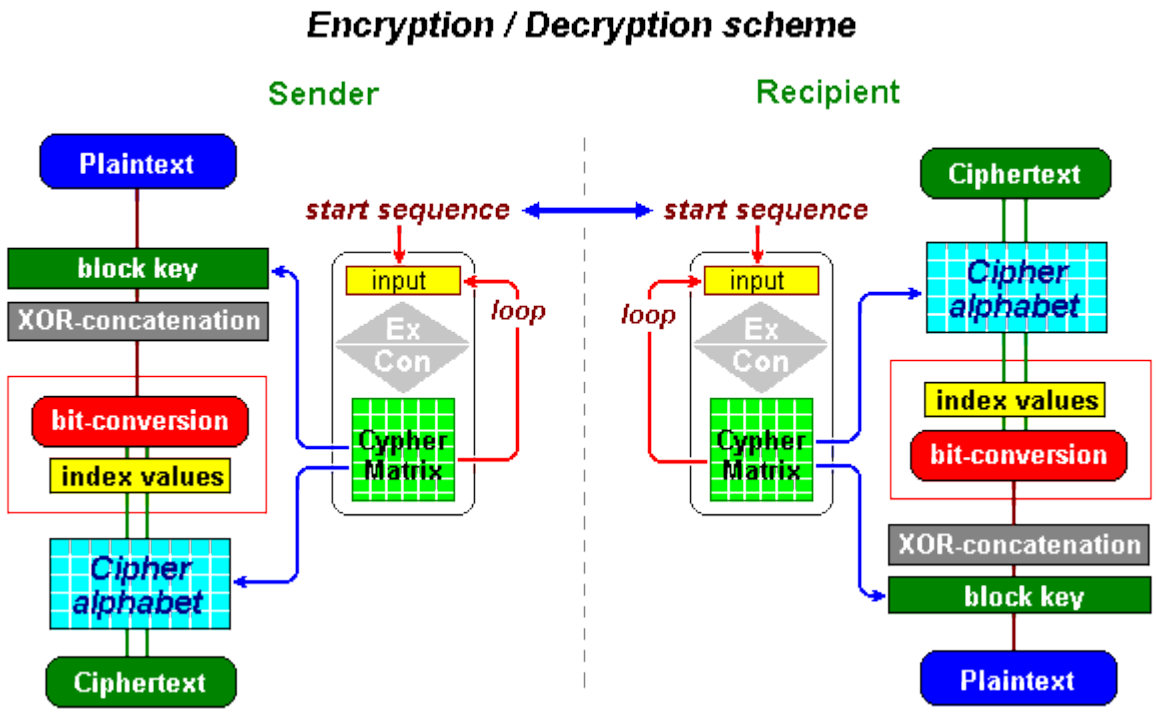
B. Verschlüsselung im Codierbereich

Die Verschlüsselung, sowohl beim Sender als auch beim Empfänger, wird in allen Runden vollständig durch die vom Generator gelieferten Steuerungsparameter **Alpha, Beta** und **Gamma** gesteuert.

Die Verschlüsselung vollzieht sich in drei Funktionen:

1. Partielles dynamisches „One-time-pad“
   Klartext-Block → Block-Schlüssel → 8-bit XOR-Verknüpfung
2. Bit-Konversion
   8-bit XOR-Verknüpfung → 7 bit Indexwerte (0 ...127)
3. Bestimmung des Chiffretexts
   7-bit Indexwerte → Chiffre-Alphabet (0...127) → Chiffretext.

Das folgende Bild zeigt die Zusammenhänge:



In jeder Runde werden der Klartext-Block (hier gewählt mit 63 Bytes) und der aus der Runden CypherMatrix entnommene gleich lange Block-Schlüssel miteinander **XOR**-verknüpft. Im Prinzip entsteht so ein partielles „**One-time-pad**“. Der Schlüssel wird auch nicht wiederholt, da in jeder Runde ein anderer Schlüssel erzeugt wird.

Im Folgenden wird der Ablauf der Verschlüsselung anhand der Textdatei: HESSE.TXT (742 Bytes) demonstriert:

> ***Als Siddhartha den Hain verließ, in welchem der Buddha, der Vollendete, zurückblieb, in welchem Govinda zurückblieb, da fühlte er, dass in diesem Hain auch sein bisheriges Leben hinter ihm zurückblieb und sich von ihm trennte. Dieser Empfindung, die ihn ganz erfüllte, sann er im langsamen***

Dahingehen nach. Tief sann er nach, wie durch ein tiefes Wasser ließ er sich bis auf den Boden dieser Empfindung hinab, bis dahin, wo die Ursachen ruhen, denn Ursachen erkennen so schien ihm, das eben ist Denken, und dadurch allein werden Empfindungen zu Erkenntnissen und gehen nicht verloren, sondern werden wesenhaft und beginnen auszustrahlen, was in ihnen ist.

Hermann Hesse, Siddhartha, Eine indische Dichtung, Montagnola 1953

Mit Eingabe der Startsequenz "**Die Schildbürger fegen den Teutoburgerwald**" ergibt sich in der ersten Runde mit den im Abschnitt **A** generierten Steuerungsdaten folgender Verlauf:

1. XOR-Verknüpfung

Die XOR-Verknüpfung als erster Schritt vollzieht sich wie folgt:

Klartext (63 Bytes)

Als Siddhartha den Hain verließ, in welchem der Buddha, der Vol

```
41 6C 73 20 53 69 64 64 68 61 72 74 68 61 20 64 65 6E 20 48 61
69 6E 20 76 65 72 6C 69 65 E1 2C 20 69 6E 20 77 65 6C 63 68 65
6D 20 64 65 72 20 42 75 64 64 68 61 2C 20 64 65 72 20 56 6F 6C
```

Block-Schlüssel (63 Bytes)

```
t-##©PÏÉÒ€□#_¡¥„|þ8óÅˆ¾½¼õò5oIdâã¶°@ö¹c™ÖG=ßYžw##%A&¤v¨1·m" yLÜ
```

```
74 2D 17 12 A9 50 CF C9 D2 80 81 00 5F A1 A5 84 7C FE 38 F3 C5
88 BE BD BC F5 F2 35 6F 49 64 E2 E3 B6 B0 40 F6 B9 63 99 D6 47
3D DF 59 9E 77 14 16 25 41 26 A4 76 A8 31 B7 6D 94 09 79 4C DC
```

XOR-Verknüpfung (63 Bytes)

```
5Ad2ú9«°áót7À…à#□#»¤áÐ□Ê□€Y#,…ÎÃßÞ`□Ü#ú¾"P□=û#4TP%BÌ#„#Ó#æ)/#°
```

```
35 41 64 32 FA 39 AB AD BA E1 F3 74 37 C0 85 E0 19 90 18 BB A4
E1 D0 9D CA 90 80 59 06 2C 85 CE C3 DF DE 60 81 DC 0F FA BE 22
50 7F 3D FB 05 34 54 50 25 42 CC 17 84 11 D3 08 E6 29 2F 23 B0
```

2. Bit-Konversion

Der zweite Schritt, die **Bit-Konversion** ist datentechnisch eine Änderung in der Anzahl der Bits. In analytischer Betrachtung vollzieht sich eine Umwandlung von Zeichen im Bytesystem zur Basis 8 (**8 Bit**) in Zeichen im Bytesystem zur Basis 7 (**7 Bi**t) [#2].
Die Folge der 8-bit Sequenzen aus der XOR-Verknüpfung teilt das Verfahren in Abschnitte von 7-bit Sequenzen. Dabei bleiben die Anzahl der Bits und ihre Reihenfolge gleich. Kein Bit wird hinzugefügt und kein Bit wird weggelassen. Die dezimalen Werte der 7-bit Sequenzen sind im nächsten Schritt Indexwerte für die Positionen der Zeichen im Chiffre-Alphabet. Die Indexwerte müssen um **+1** erhöht werden, da das Chiffretext-Alphabet als Array nur die Indexwerte **1 – 128** annimmt.

Bit-Konversion

Wandlung 8-bit XOR-Sequenzen zu 7-bit Sequenzen (Indizes:0-127)

8-bit:

```
00110101 01000001 01100100 00110010 11111010 00111001 10101011 10101101
10111010 11100001 11110011 01110100 00110111 11000000 10000101 11100000
00011001 10010000 00011000 10111011 10100100 11100001 11010000 10011101
11001010 10010000 10000000 01011001 00000110 00101100 10000101 11001110
........ .......
```

7-bit:

```
0011010 1010000 0101100 1000011 0010111 1101000 1110011 0101011 1010110
1101110 1011100 0011111 0011011 1010000 1101111 1000000 1000010 1111000
0000011 0011001 0000000 1100010 1110111 0100100 1110000 1110100 0010011
1011100 1010100 1000010 0000000 1011001 0000011 0001011 0010000 1011100
1110... ....... .......
```

Hexadezimale 7-bit Sequenzen als Index-Werte (0-127)

```
1A 50 2C 43 17 68 73 2B 56 6E 5C 1F 1B 50 6F 40 42 78 03 19 00 62
77 24 70 .........
```

Dezimale Index-Werte (+1) für das Chiffretext-Alphabet

```
27 81 45 68 24 105 116 44 87 111 93 32 28 81 112 65 67 121 4 26 1
99 120 37 113 ..........
```

Verschlüsselter Chiffretext

```
7E 72 6A C7 78 BD 40 30 A9 49 81 36 C1 72 64 EF 86 D6
34 96 C2 FE 99 AD E2 F6 8B 81 74 86 C2 C9 34 B8 C8 81
63 2F 8D A1 B5 66 EA EF 41 F5 9E 23 69 8D BE A5 CF 42
BD 74 69 6E 69 6A 84 A1 73 82 BC 86 EC FE CC AA 75 B5
.. .. .. ..
```

~rjÇx½@0©I□6Árdï†Ö4–Âþ™âö‹□t†ÂÉ4¸È□c/□¡µfêïAõž#i□¾¥ÏB½tinij„¡s,¼†ìþÌªuµ
Dâ(Ïøïv@Ÿ˜Æ}£rÓ4ùîÔ30ÈÁ%c4'é3þû=|W ‰ëÂ|ó~ÁÖÀý£
¢Á~Ú')•2òÖTŸŸþÓy˜¾ÆÚ□□'r=~

Als Folge der **„Bit-Konversion“** zeigt sich eine grundsätzliche Wirkung des Verfahrens: Die Länge des Chiffretextes verlängert sich im Verhältnis zum Klartext im Verhältnis **7:8.** Aus ursprünglich 7 Klartext-Zeichen werden mit der Konversion 8 umgewandelte Chiffretext-Zeichen. Damit kann die Forderung nicht mehr erfüllt werden, Klartext und Chiffretext sollten gleiche Länge haben. Auf jedes Klartext-Zeichen entfallen 8/7 Chiffretext-Zeichen. Gleichwohl sind Programme entwickelt, die trotz Anwendung des CypherMatrix Verfahrens zu gleicher Länge zwischen Klartext und Chiffretext führen: Speicherplatz Konvergenz.

C. Die Entschlüsselung

Die Entschlüsselung geht den umgekehrten Weg der Verschlüsselung. Die Generierung der Steuerungsparameter geschieht wie in **Abschnitt A** beschrieben. Damit entsteht ein inhaltsgleicher Ablauf wie bei der Verschlüsselung, nur in umgekehrter Reihenfolge.

1. Analyse des des Chiffretextes
   Chiffretext → Chiffre-Alphabet (0...127) → 7 bit Indexwerte
2. Bit-Konversion
   7 bit Indexwerte (0 ...127) → 8-bit XOR-Verknüpfung
3. XOR-Verknüpfung
   8-bit XOR-Verknüpfung → Block-Schlüssel → Klartext-Block

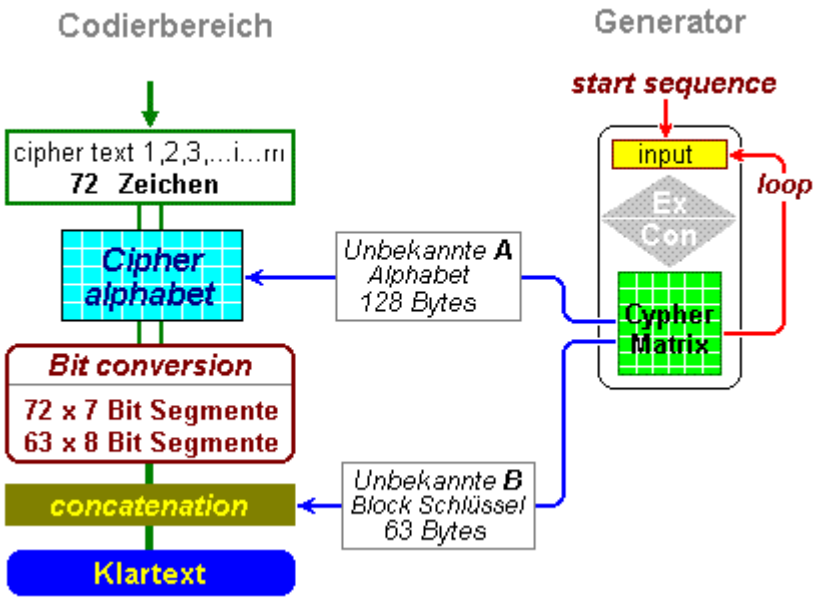
Aus Blöcken von **72 Bytes** Chiffretext sucht das Verfahren im identisch erzeugten Chiffre-Alphabet die Index-Werte der einzelnen Zeichen und verbindet ihre 7- bit Sequenzen zu einer Bitfolge von **504** Bits, die wiederum in **63** 8-bit Sequenzen aufgeteilt wird. Der entstehende String wird mit dem entsprechenden **Block-Schlüssel** XOR-verknüpft, so dass als Ergebnis der ursprüngliche Klartext wieder erscheint.

D. Sicherheit des Verfahrens

Die heute praktizierten Angriffe auf verschlüsselte Nachrichten setzen fast alle voraus, dass Klartext und Chiffretext die gleiche Länge haben. Nur dann könnten die üblichen Angriffe – Strukturanalysen, known plaintext attack, chosen plaintext attack, differenzielle und lineare Kryptoanalyse u.a - Erfolg haben. Da aber bei CypherMatrix Verschlüsselungen keine **Längenidentität** gegeben ist, sind diese Angriffe aussichtslos. Die „Bit-Konversion“ durchbricht die funktionale Beziehung zwischen Chiffretext und Klartext. Sogar **„brute force“** kann nicht greifen.

Einem Angreifer sind grundsätzlich nur der **Chiffretext** und das **CypherMatrix** Verfahren bekannt. Das jeweilige Programm und die einzelnen Steuerungsparameter, einschließlich der **Startsequenz**, kennt er nicht. Er könnte also nur eine Iteration aller Möglichkeiten versuchen. Bei einem Angriff auf die Startsequenz mit **42 Bytes** Länge ergibt sich eine Entropie von **336** und eine exponentielle Komplexität von **O(2^336)** = 1.4E+101. Ein „brute force“ Angriff ist hier offensichtlich aussichtslos.

Die Entschlüsselung geschieht in jedem Durchgang wie folgt:



Das Verfahren enthält drei Funktionen:

1. Klartextblock --> **Block-Schlüssel** --> -8 bit XOR-Sequenzen
2. 8-bit XOR Sequenzen --> 7-bit Index-Werte
3. 7-bit Index-Werte --> **Chiffre-Alphabet (128)** --> Chiffretext

In diesen Funktionen sind die Parameter **Block-Schlüssel** und **Chiffre-Alphabet** zwei voneinander unabhängige Variable. Es gelten:

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

fx = funktionale Verbindung
pn = Klartext
k1 = **Block-Schlüssel**
b1 = 8-bit Sequenz
b2 = 7-bit Index-Wert
k2 = **Chiffre-Alphabet (128)**
cm = Chiffretext

Die Ermittlung des Chiffretextes **cm** und die retrograde Suche nach dem Klartext **pn** zeigen sich somit als Gleichungen mit zwei unbekannten Veränderlichen: **k1** und **k2**. Das führt bekanntlich nur dann zu einer eindeutigen Lösung, wenn eine Unbekannte aus der anderen abgeleitet werden kann oder wenn zwei Gleichungen mit denselben Unbekannten vorhanden sind.

Aber zwischen dem jeweiligen Block-Schlüssel = k1 und dem in derselben Runde generierten Chiffre-Alphabet (128) = k2 gibt es keine Verbindung. Beide sind zwar aus der aktuellen CypherMatrix entnommen, haben aber keine funktionale Beziehung: (k1 --> (**Hk MOD 169**)**+1**) und k2 --> (**Hk +Hp**) **MOD 255**)**+1**). Die Runden CypherMatrix selbst ist aus der ursprünglichen Start-Sequenz hergeleitet. Dahin führt jedoch kein Weg zurück (zwei Einwegfunktionen stehen dagegen). Es gibt somit viele Paare **Chiffre-Alphabet** / **Block-Schlüssel**, die nach einem versuchten "brute force" Angriff irgendwelche lesbaren Texte liefern, von denen man aber nicht weiß, welcher der Richtige ist: Angriff mit "brute force" [#3].

D. Kombinierte Operationen

Das nach den vorstehenden Erläuterungen aufgebaute Verschlüsselungsprogramm ist das einfachere seiner Art. Die Operationen XOR-Verknüpfung und Bit-Konversion können mit weiteren Kombinationen ergänzt oder ersetzt werden (beispielsweise: „dyn24“, „bit exchange“, „bit crossing“, Byte-Transposition u.a.). Der Vielfalt sind keine Grenzen gesetzt. Einzelheiten sind im Artikel: "Der Kern des CypherMatrix Verfahrens" nachzulesen.

München, im Juni 2010

[#1] Patent des Autors: DPMA 19811593 vom 18.03.1998
[#2] Bit- und Byte-Technik in der Kryptographie
[#3] Schneier, Bruce, Angewandte Kryptographie (deutsche Ausgabe) Bonn, S. 17