

# „CypherMatrix“ Eine Basisfunktion in der Kryptographie

(Ernst Erich Schnoor)

„CypherMatrix“ ist ein neues kryptographisches Verfahren, das als Basisfunktion in fast allen Bereichen der Informationstechnik eingesetzt werden kann.

## A. Systematisierung der Bitfolgen

Die vom Computer generierten Bitfolgen sind technisch unbegrenzt. Um mit ihnen systematisch arbeiten zu können, müssen sie **skaliert** und in definierte **Abschnitte** (Einheiten) geteilt werden. Bisher dienen diesem Ziel nur die Begriffe „nibble“, „word“ und Byte (als eine Folge von 8 Bits). Aber für eine systematische Aufarbeitung digitaler Informationen ist das zu wenig und augenscheinlich auch unvollständig.

Alle uniformen Bitlängen, wie z.B: 4-bit, 7-bit, 8-bit, 12-bit, 16-bit, 32-bit u.a. können als eigene Einheiten „**Units**“ (8-bit Folgen = **Bytes**) definiert werden. Dabei bietet sich ein Strukturvergleich an mit Begriffen aus der Zahlentheorie. Das Bit entspricht der Ziffer, das Byte der Zahl und das Zeichenalphabet der geordneten Menge. Entsprechend der Zahlentheorie kann dann für 8-bit Folgen beispielsweise vom „**Bitsystem zur Basis 8**“ gesprochen werden.

	Alphabet
Bitfolgen: 1-bit = Bitsystem zur Basis 1 = $2^1$ Zeichen =	2 Units
2-bit = Bitsystem zur Basis 2 = $2^2$ Zeichen =	4 Units
7-bit = Bitsystem zur Basis 7 = $2^7$ Zeichen =	128 Units
8-bit = Bitsystem zur Basis 8 = $2^8$ Bytes =	256 Bytes
12-bit = Bitsystem zur Basis 12 = $2^{12}$ Zeichen =	4096 Units
16-bit = Bitsystem zur Basis 16 = $2^{16}$ Zeichen =	65536 Units
32-bit = Bitsystem zur Basis 32 = $2^{32}$ Zeichen =	4294967296 Units
.....	.....

Die einzelnen Alphabete sind völlig unabhängig voneinander. Schwierigkeiten bereitet es nur, eigene Symbole für die Zeichen (Units) im jeweiligen Bitsystem zu definieren. Für das Bitsystem zur Basis 1 bestehen die Zeichen „0“ und „1“, historisch: „L“ und „H“. Das Bitsystem zur Basis 4 kennt die Einheit „nibble“ und 16-bit im Bitsystem zur Basis 16 sind „1 word“.

## B. CypherMatrix als Basisfunktion

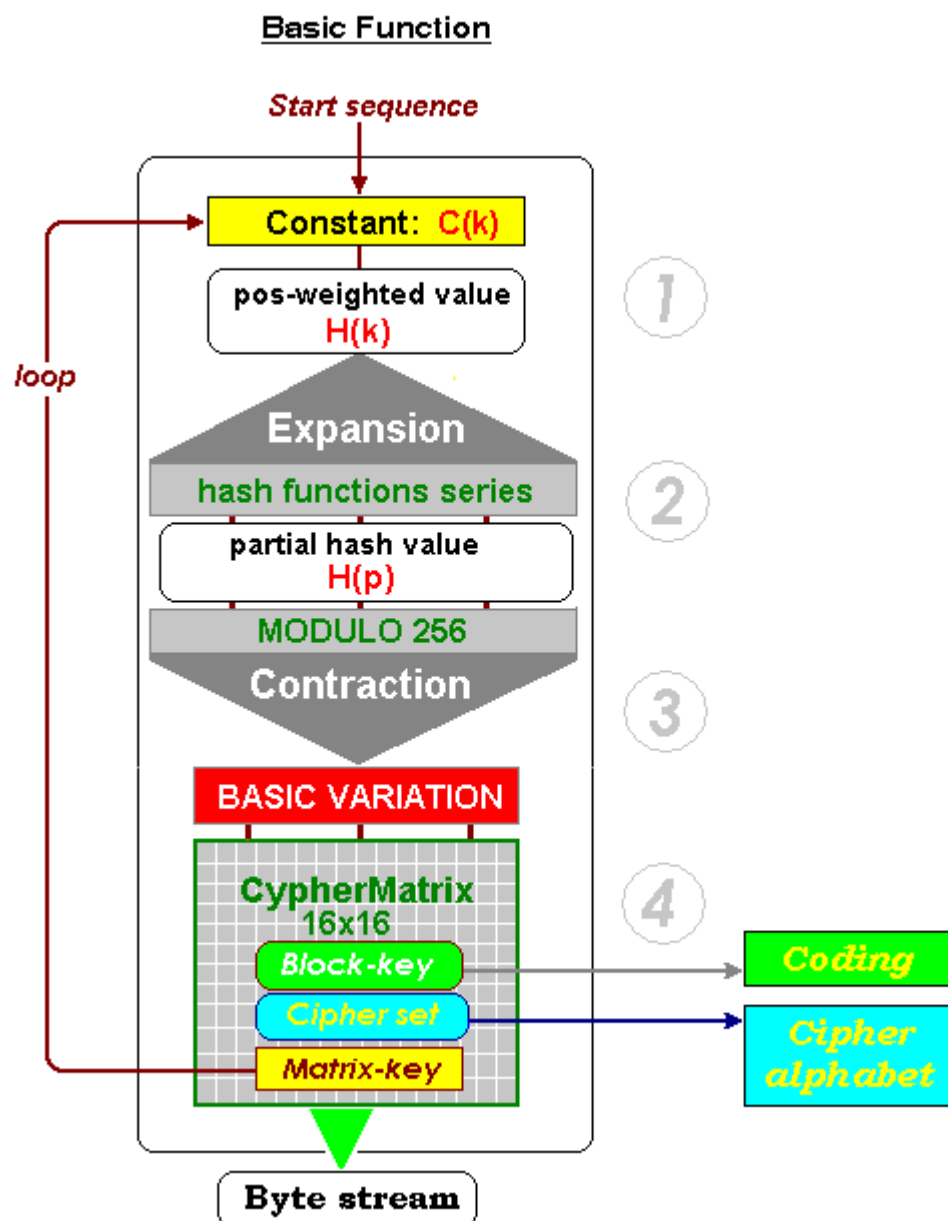
Die Funktion beginnt mit der Eingabe einer Information als **Start Sequenz** und führt die Information zu einer eindeutigen und kollisionsfreien Abbildung als **CypherMatrix** mit 16x16 Elementen.

## 1. Verlauf der Funktion

Die Eingangssequenz wird positionsgewichtet, mit einer Konstanten **Ck** multipliziert, zu einer Hashfunktionsfolge erweitert (**Expansion**), wieder zur BASIS VARIATION verdichtet (**Contraction**) und mit einer dreifachen Permutation zur finalen CypherMatrix entwickelt. Die CypherMatrix liefert die für kryptographische Lösungen erforderlichen Steuerungsparameter (Chiffre-Alphabet, Block-Schlüssel, Matrix-Key als Eingangssequenz für den nächsten Durchlauf und weitere Faktoren). In jeder Runde werden die CypherMatrix mit 16x16 Elementen neu generiert und die Steuerungsparameter neu berechnet

Nach den Gesetzen der Wahrscheinlichkeit entsteht eine Wiederholung der gleichen CypherMatrix erst in **256!** (Fakultät) = **8E+506** Fällen.

Die folgende Skizze veranschaulicht die Zusammenhänge:



Als Information wählen wir folgende Startsequenz:

**"Bruno der Braunbär aus Bregenz im Breisgau"** (n = 42).

42 72 75 6E 6F 20 64 65 72 20 42 72 61 75 6E 62 84 72 20 61 75  
73 20 42 72 65 67 65 6E 7A 20 69 6D 20 42 72 65 69 73 67 61 75

Die Startsequenz ist eine Folge **m** bestimmter Bytes **a(i)** mit der Länge (**n**): z.B. Nachricht, Passphrase, Blockschlüssel, Pixel, Klänge usw. Um die Folge als Sachverhalt zu analysieren, muss sie systematisiert (skaliert) werden. Dazu wird jedem Byte **a(i)** ein Index zugeordnet und alle (**n**) Bytes werden in sachgerechter Weise miteinander verknüpft.

$$m = a_1 + a_2 + a_3 + \dots a_i + \dots a_n$$

(Der einzelne Wert für "a" wird um (+1) erhöht da sonst ASCII-null (0) nicht berücksichtigt wird)

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

$$m = 3993$$

Um die einzelnen Bytes **a(i)** innerhalb der Zeichenfolge zu unterscheiden, müssen weitere Merkmale hinzukommen, da anderenfalls kein eindeutigen Ergebnisse erzielt werden können.



## Erweiterung der **Startsequenz** zum **positionsgewichteten Zwischen-Wert H<sub>k</sub>**

Mit Besinnung auf **Renè Descartes** (1596 – 1650) wissen wir, dass jeder Sachverhalt - soweit er in seinen Dimensionen skalierbar ist – durch seine Koordinaten für **Gegenstand**, **Ort** und **Zeit** (kartesisches Koordinatensystem) eindeutig bestimmt wird. Wir definieren:

Sachverhalt = (**m**) digitale Information der Länge (**n**)  
Gegenstand = **a(i)** Element der Information, Zeichen, Byte  
Ort = **p(i)** Position von **a(i)** innerhalb der Information  
Zeit = **t(i)** Zeitpunkt von **a(i)** innerhalb der Information

Damit sich die einzelnen Zeichen unterscheiden wird jedes Byte **a(i)** mit seinem Ort **p(i)** multipliziert, d.h. **positionsgewichtet**. Die Zeit ist nur dann von Bedeutung, wenn zwischen den einzelnen Bytes und der Prozessorfrequenz eine variable Funktion besteht, ansonsten: **t(i) = 1** .

Um einen bestimmten Wert für die Folge **m** zu erhalten, werden die positionsgewichteten Dimensionswerte zum Wert **H(k)** addiert.

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

$$H(k) = 86560$$

Mit der **Positionsgewichtung** unterscheiden sich zwar die Bytes **a(i)**, aber Kollisionen als Folge des Austausches von Bytes innerhalb der Information sind noch nicht ausgeschlossen.

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

Um Kollisionen zu vermeiden, wird die Positionsgewichtung in einen Bereich oberhalb der Länge (**n**) der Zeichenfolge verschoben, indem **p(i)** um einen konstanten Abstand erweitert wird. Der Abstand, mit Konstante **C(k)** bezeichnet, erfüllt die Aufgabe, Kollisionen zu vermeiden. Ausführliche Erläuterungen sind im Artikel: ["Bestimmungsfaktoren für Kollisionsfreiheit"](#) [#1] dargelegt.

Mit Code – eine gewählte Zahl zwischen 1 und 99 - kann die Funktion individualisiert werden. Wir setzen: **code = 1**

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

$$C(k) = 1681$$

Nach Einbindung der Konstanten **C(k)** wird der Zwischenwert wie folgt ermittelt:

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

$$H_k = 6798793$$

Basis 16: <b>67BDC9</b>	Basis 62: <b>SWfx</b>
Basis 128: <b>3U °Φ</b>	Basis 256: <b>ô Ĩ ¶  </b>

Das Ergebnis **H(k)** vermeidet Kollisionen, ist aber immer noch zu niedrig, um einen sicheren und unangreifbaren Wert zu begründen. Es kann höchstens als **MAC** für Nachrichten dienen.



## Hochrechnung der **Startsequenz** zur Hash-Funktionsfolge im Zahlensystem zur **Basis 77 (Expansion)** und Hashwert **Hp**

Um einen sicheren und eindeutigen Wert zu erreichen, wird eine „**Expansionsfunktion**“ eingeführt, die die Bestimmungsfaktoren auf möglichst viele Variablen erweitert (etwa 160 bis 2400 Zeichen). Hierzu wird der Wert jedes Zeichens **a(i)** mit der Position **p(i)** und dem Zwischenwert **H(k)** verknüpft (multipliziert) und zu einem

weiteren Zwischenergebnis hochgerechnet. Das jeweilige dezimale Zwischenergebnis **s(i)** wandelt das Verfahren um in **d(i)**, einem Wert in einem höheren Zahlensystem (z.B. zur **Basis 77**). Die Ziffern **d(i)** werden seriell in einer fortlaufenden Folge gespeichert (hier: **Hash Funktionsfolge** genannt). In jeder Runde **r** (**r**<sub>1</sub>,...**r**<sub>j</sub>, ...**r**<sub>m</sub>) wird gleichzeitig der finale Bestimmungswert **H(p)** als Summe aller Werte **s(i)** errechnet.

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

$$S_i \text{ ---> } d_i \text{ (Basis 77)}$$

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

$$H(p) = 588503523067$$

Das gewählte Zahlensystem zur **Basis 77** umfasst die folgenden Ziffern:

0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ  
 abcdefghijklmnopqrstuvwxyz&#@àáâãäåæçèéêë  
 (definiert vom Autor, nicht standardisiert)

In Ausschnitten zeigt sich die Berechnung der Ziffern der **Hash Funktionsfolge** wie folgt:

Zchn	pi	Hk	(ai+1)*pi*Hk	Si	Basis 77			
(ai+1)	(ai+1)*pi		pi+code+r					
B	67	11	737	6798793	5010710441	13	5010710454	1âfhêv
r	115	12	1380	6798793	9382334340	14	9382334354	3Zäl28
a	98	13	1274	6798793	8661662282	15	8661662297	3FUrvp
u	118	14	1652	6798793	11231606036	16	11231606052	4Bcè#p
n	111	15	1665	6798793	11319990345	17	11319990362	4E1gçà
b	99	16	1584	6798793	10769288112	18	10769288130	3êRM9&
ä	133	17	2261	6798793	15372070973	19	15372070992	5qMP1I
r	115	18	2070	6798793	14073501510	20	14073501530	5FQâ3B
..	...	..	.....	.....	.....	..	.....	.....
					Summe H(p)		588503523067	2#WECDQ

In unserem Beispiel enthält die **Hash Funktionsfolge** 247 Ziffern im Zahlensystem zur Basis 77:

CèxëäibGQ3ãZàOp18âBYà1VNn#ScMoãq1xuzçN23#4Uu2kDZ1j##XbY1âfhêv3Zäl283FU  
 rpv4Bcè#p4E1gçà3êRM9&5qMP1I5FQâ3B1iKW#W4æ5wâv6HJçlb6VhèBZ1âzFWE42ërèH7  
 H3Btr6oãEM&746Stà7DSFUb86i62G9KpOWQ2hàvkt8e2XTc994yPk2#06435âfKççAUr#6  
 K9aãKQeA92äd@BRéXYMAYhp8iA77ècvCYdâaç

Die Variablen sind Ziffern (keine Zeichen) im Zahlensystem zur **Basis 77**.  
 Es gibt keinen Weg zurück zur Startsequenz (erste Einweg-Funktion).

Gleichzeitig errechnet das Programm die folgenden Bestimmungsfaktoren:

	dezimal	Basis 77
Hashkonstante C(k):	<b>1681</b>	<b>L@</b>
Positionsgewichteter Wert (H <sub>k</sub> ):	<b>6798793</b>	<b>ETπS1</b>
Zwischenwert (H <sub>p</sub> ):	<b>588503523067</b>	<b>2#WECDΩ</b>
Gesamtwert (H <sub>p</sub> +H <sub>k</sub> ):	<b>588510321860</b>	<b>2#WT3Γδ</b>

Aus den Bestimmungsfaktoren werden folgende Parameter abgeleitet:

<b>Variante</b>	$(H_k \text{ MOD } 11) + 1$	=	<b>2</b>	Beginn der Kontraktion
<b>Alpha</b>	$((H_k + H_p) \text{ MOD } 255) + 1$	=	<b>36</b>	Offset Chiffre-Alphabet
<b>Beta</b>	$(H_k \text{ MOD } 169) + 1$	=	<b>93</b>	Offset Block-Schlüssel
<b>Gamma</b>	$((H_p + \text{code}) \text{ MOD } 196) + 1$	=	<b>49</b>	Offset Matrix-Schlüssel
<b>Delta</b>	$((H_k + H_p) \text{ MOD } 155) + \text{code}$	=	<b>31</b>	dynamische Bitfolgen
<b>Theta</b>	$(H_k \text{ MOD } 32) + 1$	=	<b>10</b>	dynamische Zahlenfolgen

3

### Verdichtung der Hash-Funktionsfolge durch Modulo 256 zum Array BASIS VARIATION (Contraction)

Um die Variablen auf dezimale Größen zurückzuführen wird als Nächstes eine **Kontraktionsfunktion** (zweite Einweg-Funktion) eingeführt. Für die Ziffern der **Hash Funktionsfolge** wird das Zahlensystem zur **Basis 78** (expansion base +1) unterstellt. Jeweils drei Ziffern der Funktionsfolge werden seriell durch **MODULO 256** in dezimale Zahlen 0 bis 255 (ohne Wiederholung) zurückgerechnet. Das Ergebnis wird in einem Array von 16x16 Elementen **BASIS VARIATION** gespeichert. Eine rückwärts gerichtete Bestimmung vorhergehender Daten ist nicht möglich.

Die Rückrechnung auf dezimale Zahlen durch MODULO 256 beginnt beim Parameter **Variante = 2**:

. èxëåibGQ3.....

Ziffern Basis 78	dezimal	MODULO 256	- Theta	Element
<b>èxë</b>	448810	42	10	<b>32</b>
<b>xëå</b>	364954	154	10	<b>144</b>
<b>ëåi</b>	467888	176	10	<b>166</b>
<b>åib</b>	429349	37	10	<b>27</b>
<b>ibG</b>	270598	6	10	<b>252</b> (256 + (6-10))
<b>bGQ</b>	226382	78	10	<b>68</b>
<b>GQ3</b>	99375	47	10	<b>37</b>

### BASIS-VARIATION (256 Elemente) Verteilung der Elemente

**032 144 166 027 252 068 037** 012 029 241 168 061 225 088 109 139  
190 255 205 039 067 229 191 200 224 016 244 078 042 223 079 236

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

Die Werte der **BASIS VARIATION** werden direkt auf Indexwerte von Bytes bezogen mit Werten von **0** bis **255** (vergleichbar: ASCII-Zeichensatz). Auf diese Weise generiert das Verfahren aus allen Elementen der BASIS VARIATION in jeder Runde die erste „CypherMatrix“ mit 16x16 Elementen.



## Permutation der **BASIS VARIATION** zur CypherMatrix als finalen **Hashwert H**

Die erste „CypherMatrix“ (**CM1**) zeigt sich wie folgt:

```

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

Durch dreifache Permutation (Version B) entsteht die finale „CypherMatrix“. Im Pseudo-Code geschieht folgendes:

```

For s = 1 TO 3
  For i = 1 TO 16
    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$(3,i,a) = Matrix$(2,i,j)
CASE 3
        CM3Set$ = CM3Set$ + Matrix$(3,i,j)  CypherString
END SELECT
    Next j
Next i
Next s

```

### Finale **CypherMatrix** als Ergebnis der dreifachen Permutation (256 Elemente)

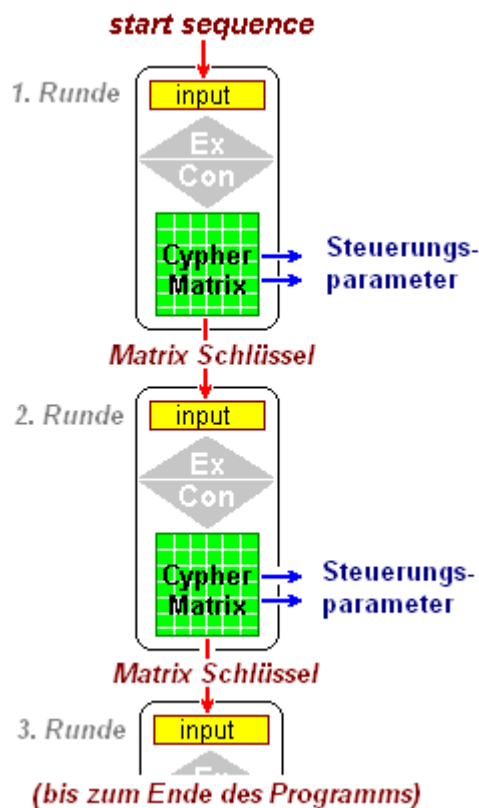
1	65	B4	BE	9F	7A	6B	02	52	72	69	D3	41	36	B3	48	A2	16
17	CB	F3	85	6F	8B	9B	6C	60	54	9C	ED	51	29	0A	DC	37	32
33	9D	EF	E7	0F	DB	EC	6D	9A	22	2F	15	D6	2C	F8	96	B7	48
49	E3	84	83	35	50	F2	AF	4F	58	99	0B	5D	7E	C7	08	92	64
65	67	94	EE	0D	CF	A4	CE	AE	3E	DF	E1	8E	07	D8	BD	38	80
81	98	B0	3A	BA	E8	C2	7F	76	26	33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

Die **CypherMatrix (CM3)** stellt eine eindeutige und kollisionsfreie Abbildung der Startsequenz dar. Im Ergebnis kann die CypherMatrix als Bestimmungsbasis für eine Vielzahl von kryptographischen Problemen verwendet werden, insbesondere wenn sie zum „Generator“ erweitert wird.

## 2. Die Basisfunktion als Generator

Die bisherigen Erläuterungen der **Basis Funktion** beziehen sich nur auf einen Durchlauf ( $r_j$ ). Um die Funktion auf einen universell einsetzbaren kryptographischen Mechanismus zu erweitern, wird jeweils eine neue Startsequenz – bezeichnet als **Matrix Key** (Folge von 42 Bytes) – aus der laufenden CypherMatrix herausgezogen und auf den Beginn der nächsten Runde zurückgeführt ("**loop**"). Als Startsequenz initialisiert der Matrix-Schlüssel den nächsten Durchgang ( $r_{j+1}$ ) und durchläuft die **Basis Funktion** erneut, so dass eine unbegrenzte Zahl von Runden entsteht, bis ein Endeimpuls gesetzt wird.

### **Generator**



Die **Start Sequenz** der ersten Runde steuert das gesamte Verfahren. Alle weiteren Eingangs-Sequenzen (**Matrix-Schlüssel**) werden vom Programm erzeugt.

Der Parameter **Gamma** bestimmt den Offset für den **Matrix-Schlüssel** zur Initialisierung der nächsten Runde. Ab der Position **49** entnimmt das Verfahren die folgende Startsequenz von **42 Bytes**:

hexadezimal:

49	E3	84	83	35	50	F2	AF	4F	58	99	0B	5D	7E	C7	08	92	64
65	67	94	EE	0D	CF	A4	CE	AE	3E	DF	E1	8E	07	D8	BD	38	80
81	98	B0	3A	BA	E8	C2	7F	76	26	33	..	..	..	..	..	..	96

ASCII-Zeichensatz:

ã„f5PòOX™#]~Ç#g”î ïαî®>βáŽ#Ø½8~°:èÂ□v&3

Alle weiteren Matrix-Schlüssel der folgenden Runden werden vom Programm bestimmt.

### 3. Sensibilität der Basisfunktion

Eine kleine Änderung der Information (Startsequenz) erzeugt eine große Wirkung in den Ergebnissen (**Avalanche effect**). Wird beispielsweise das letzte **Bit** in der Eingangssequenz von „1“ auf „0“ geändert, während alle übrigen Daten unverändert bleiben, ergeben sich die folgenden Daten:

Original: Bruno der Braunbär aus Bregenz im Breisgau  
 . Änderung: Bruno der Braunbär aus Bregenz im Breisgat

„u“ = 111 0101  
 „t“ = 111 0100

Die **Hash Funktionsfolge** umfasst 247 Ziffern im Zahlensystem zur Basis 77:

CèeeqiaQcWäYdYQ18àaWm1VLf6ZcLâR81xsAb223xêKV2k9gly#&EjC1àcy&D3Z@4ëS3FP  
 äXæ4BWtVb4DçL6x3êLN&l5qDyë55FJ7bf1il4IL4âêSMh6HAj9G6VYPeO1âwPàP42âmpl7  
 GäNHä6oyDI073çxsM7DHWwà86Væhá9KbU6k2hzäOX8dámñë98â6XZ2&ç@ëa5ãWWawAUcEê  
 39au1kBA8@qèzBRvRvYAYRÉE#A6äzàaCQBàêp

Bestimmungsfaktoren:	dezimal	Basis 77
Hashkonstante C(k):	<b>1681</b>	<b>L@</b>
Positionsgewichteter Wert (H <sub>k</sub> ):	<b>6797070</b>	<b>ΕπVn</b>
Zwischenwert (H <sub>p</sub> ):	<b>588068903247</b>	<b>2#J#C8u</b>
Gesamtwert (H <sub>p</sub> +H <sub>k</sub> ):	<b>588075700317</b>	<b>2#K13eS</b>

Parameter abgeleitet aus den Bestimmungsfaktoren:

<b>Variante</b>	(H <sub>k</sub> MOD 11) +1	=	<b>6</b>	Beginn der Kontraktion
<b>Alpha</b>	((H <sub>k</sub> + H <sub>p</sub> ) MOD 255) +1	=	<b>238</b>	Offset Chiffre-Alphabet
<b>Beta</b>	(H <sub>k</sub> MOD 169) +1	=	<b>60</b>	Offset Block-Schlüssel
<b>Gamma</b>	((H <sub>p</sub> + code) MOD 196) +1	=	<b>37</b>	Offset Matrix-Schlüssel
<b>Delta</b>	((H <sub>k</sub> + H <sub>p</sub> ) MOD 155) +code	=	<b>38</b>	dynamische Bitfolgen
<b>Theta</b>	(H <sub>k</sub> MOD 32) +1	=	<b>15</b>	dynamische Zahlenfolgen

### BASIS-VARIATION (256 Elemente)

Verteilung der Elemente

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

## Finale **CypherMatrix** als Ergebnis der dreifachen Permutation (256 Elemente)

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

Die rot gekennzeichneten Elemente (ab **Gamma = 37**) sind der Matrix-Schlüssel für die nächste Runde. Insoweit wirkt sich die Änderung auch nur eines Bits auf den Verlauf aller weiterer Runden aus, d.h. des gesamten Programms.

Die Änderung um ein **Bit** am Ende der Eingabesequenz führt zu folgenden Änderungen in den Bestimmungsfaktoren:

	Positions- gewichteter Wert ( $H_k$ )	Zwischenwert ( $H_p$ )	Gesamtwert ( $H_p+H_k$ )
Original:	6798793	588503523067	588510321860
Änderung:	<u>6797070</u>	<u>588068903247</u>	<u>588075700317</u>
Differenz:	1723	434619820	434621543

Die weiteren Änderungen zeigen sich in der **BASIS-VARIATION** und in der finalen **CypherMatrix**.

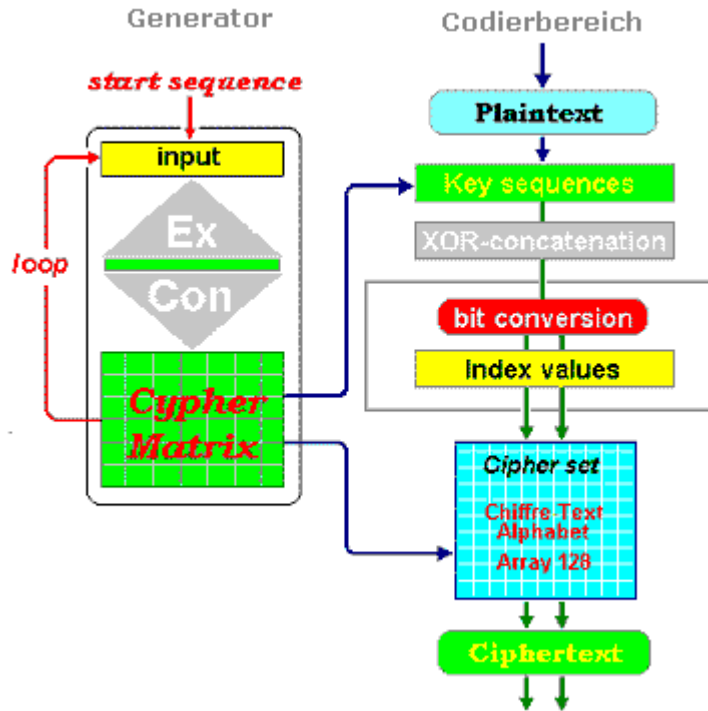
### C. Anwendungen der Basisfunktion

Die Basisfunktion „**CypherMatrix**“ entwickelt sich zu einem allgemeinen kryptographischen Werkzeug, das in fast allen Gebieten der Kryptographie eingesetzt werden kann. Sie liefert alle Steuerungsparameter für kryptographische Anwendungen (z.B. Matrix-Schlüssel, Chiffre-Alphabete, Block-Schlüssel, Bytefolgen, Hashwerte, Authentifizierungs Merkmale, S-Boxen usw.), insbesondere jedoch für:

- Durchführung von Verschlüsselungen,
- Berechnung von Hashwerten,
- einfache und erweiterte Signaturen sowie
- Generierung unbegrenzter Byte- und Zahlenfolgen

# 1. Verschlüsselungen

Im Gegensatz zu den heute üblicherweise verwendeten Algorithmen gehen die CypherMatrix Verschlüsselungen eigene Wege.



Die Verschlüsselung wird auf zwei Bereiche verteilt. Beide Bereiche werden kombiniert, können aber auch getrennt verwendet werden. Aufgabe der **Basisfunktion** (Generator) ist die Bereitstellung von Steuerungsparameter für den Ablauf des Verfahrens. Die eigentliche **Verschlüsselung** vollzieht sich im Codierbereich.

## a) Startsequenz zur Steuerung des Generators

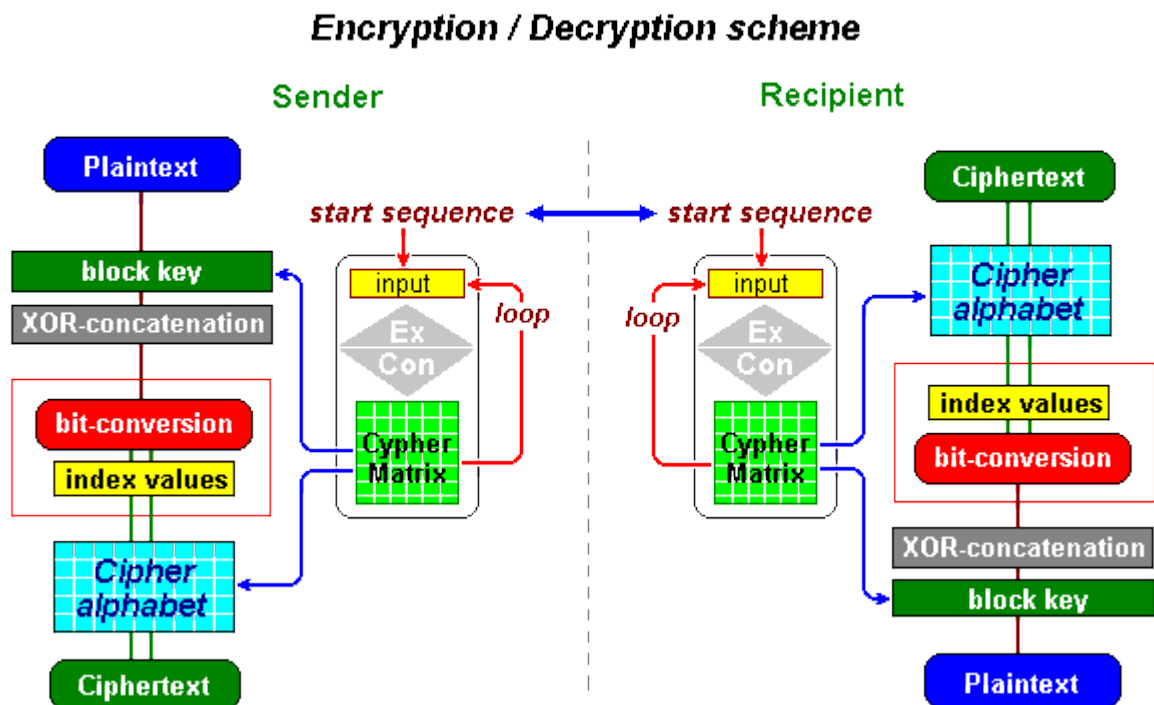
Das Verfahren ist symmetrisch, weil Sender und Empfänger zur Initialisierung des Generators die gleiche Startsequenz eingeben müssen. Das Verfahren ist dynamisch, weil der Generator in jeder Runde für jeden Block (z.B. 63 Bytes) neue unabhängige Steuerungsparameter generiert.

Die Startsequenz (Passphrase) sollte ungewöhnlich sein und dennoch leicht zu behalten, so dass sie nicht aufgeschrieben werden muss aber auch nicht geraten werden kann. Einige Beispiele:

Die Schildbürger fegen den Teutoburgerwald	[42 Bytes]
Die weiße Elster fließt in das schwarze Meer	[44 Bytes]
9 kangaroos jumping along the Times Square	[42 Bytes]
Blue flamingos flying to Northern Sutherland	[44 Bytes]

Wegen ihrer Länge kann die Startsequenz 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.

Das folgende Schema zeigt die Zusammenhänge:



Mit der Eingabe der gleichen Startsequenz, sowohl beim Sender als auch beim Empfänger, werden im gesamten Verfahren ein identischer Verlauf und identische Steuerungsparameter erzeugt.

## b) Dynamisches „one-time-pad“

Für den ersten Verschlüsselungsschritt in jedem Durchlauf generiert das Verfahren einen internen Block-Schlüssel in der gleichen Länge wie der eingelesene Klartextblock (z.B: **63 Bytes**) und verknüpft beide mit der **XOR**-Funktion. Auf diese Weise entsteht ein partielles „one-time-pad“. Klartext und Schlüssel sind stets gleich lang. Der Schlüssel wird auch nicht wiederholt, da in jeder Runde ein anderer Schlüssel aus der aktuellen CypherMatrix entnommen wird.

Das Ergebnis der XOR-Verknüpfung (**63 Bytes**) wird mit einer „**Bit-Konversion**“ vom Bitsystem zur Basis **8** in Zeichen des Bitsystems zur Basis **7** (**72 Zeichen**) umgewandelt. Das zugehörige **Alphabet** von 128 Chiffre-Zeichen entnimmt das Programm aus der aktuellen CypherMatrix.

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**.

Im Folgenden wird der Ablauf der Verschlüsselung anhand der Textdatei: “**Fontane.txt**” (432 Bytes) gezeigt. Als Startsequenz dient die im vorhergehenden Abschnitt gewählte Passphrase:

**"Bruno der Braunbär aus Bregenz im Breisgau"** (n = 42).

Der Text der Datei lautet:

*Herr von Ribbeck auf Ribbeck im Havelland,  
ein Birnbaum in seinem Garten stand.  
Und kam die goldene Herbsteszeit  
und die Birnen leuchteten weit und breit,  
da stopfte, wenn's Mittag vom Turme scholl,  
der von Ribbeck sich beide Taschen voll.  
Und kam in Pantinen ein Junge daher,  
so rief er: "Junge, willst 'ne Beer?"  
Und kam ein Mädcl, so rief er: "Lütt Dirn,  
kumm man röwer, ick hebb 'ne Birn".*

*Theodor Fontane, 1819-1898*

Der erste **Klartext-Block** [63 Bytes im Bitsystem zur Basis 8] wird eingelesen

**Herr von Ribbeck auf Ribbeck im Havelland, ein Birnbaum in sei**

48 65 72 72 20 76 6F 6E 20 52 69 62 62 65 63 6B 20 61 75 66 20  
52 69 62 62 65 63 6B 20 69 6D 20 48 61 76 65 6C 6C 61 6E 64 2C  
0D 0A 65 69 6E 20 42 69 72 6E 62 61 75 6D 20 69 6E 20 73 65 69

und mit dem ersten **Block-Schlüssel** [63 Bytes ab Offset: 93] XOR-verknüpft

=€#^Š×¼S|fÉB«‡KÝGN"}ñwž##Lp#j\##I2□ô ##utZû~úÁ#p¹—Áª@Ãà s[ÒìÐÕ

3D 80 06 5E 8A D7 BC 53 7C A3 E9 42 AB 87 4B DD 47 4E A8 7D F1  
77 9E 1F 14 4C FE 18 A1 5C 01 11 49 32 8F F4 A0 10 1D 75 74 5A  
FB AC FA C5 13 70 B9 97 C1 AA 40 C3 00 E0 0C 73 5B D2 CC D0 D5

zur **XOR Sequenz** [63 Bytes im Bitsystem zur Basis 8]

uât,ªjÓ=ñ€ Éâ(¶g/Ý#Ñ%÷)v)□s□5l1#Sù'ì||##vö;ÿ~}Pûp³Ä"çu□,#5ð¿µ¼

```

75 E5 74 2C AA A1 D3 3D 5C F1 80 20 C9 E2 28 B6 67 2F DD 1B D1
25 F7 7D 76 29 9D 73 81 35 6C 31 01 53 F9 91 CC 7C 7C 1B 10 76
F6 A6 9F AC 7D 50 FB FE B3 C4 22 A2 75 8D 2C 1A 35 F2 BF B5 BC

```

Die Bitfolgen im Ablauf:

Klartext (Basis 8):

01001000 01100101 01110010 01110010 00100000 01110110 01101111 01101110 ....

Block-Schlüssel (Basis 8):

00111101 10000000 00000110 01011110 10001010 11010111 10111100 01010011 ....

XOR-Verknüpfung (Basis 8):

01110101 11100101 01110100 00101100 10101010 10100001 11010011 00111101 ....

Die XOR-verknüpften Folgen (Bitsystem zur Basis 8) werden zu Zeichen im Bitsystem zur Basis 7 umgewandelt.

### c) Bit-Konversion

Die Umwandlung von Basis 8 in das Bitsystem zur Basis 7 (Bit-Konversion) erfolgt durch abschnittsweise Teilung der Bitfolgen. Die Folge der 8-bit Sequenzen aus der XOR-Verknüpfung teilt das Verfahren in 7-bit Abschnitte. Dabei bleiben die Anzahl der Bits und ihre Reihenfolge gleich. Kein Bit wird hinzugefügt und kein Bit wird weggelassen. Die dezimalen Werte der Zeichen im Bitsystem zur Basis 7 sind Indexwerte für die Positionen der Zeichen im Chiffre-Alphabet. Die Indizes für das Chiffre-Alphabet müssen um +1 erhöht werden, da der Index „0“ im Array des Chiffre-Alphabets nicht erkannt wird.

Das aus der aktuellen CypherMatrix entnommene **Chiffre-Alphabet** mit 128 Elementen liefert die Chiffre-Zeichen für die Werte des Bitsystems zur Basis 7:

#### Cipher Alphabet (8x16) [ab Offset 36]

1	ì m š / Ö ø - · ã „ f 5 P ò ¯ O	16
17	X ™ ] ~ Ç ' g " î ï ð ñ ® > á Ž	32
33	Ø ½ 8 ~ : ° è Å □ v & 3 • * = €	48
49	^ Š × ¼ S   £ é B « ‡ K G N " }	64
65	ñ w ž L p j \   2 □ ô u t Z ù	80
81	¬ ú Å p ¹ — Á º @ ã à s [ Ò Ì Ð	96
97	< ' ý n “ Œ ð b È % ; 4 9 J »	112
113	F U Y ÷ , ? À ^ ¿ D { . ¶ ù Ù □	128

#### Cipher Alphabet (hex base 16)

1	EC 6D 9A 2F D6 F8 96 B7 E3 84 83 35 50 F2 AF 4F	16
17	58 99 5D 7E C7 92 67 94 EE CF A4 CE AE 3E E1 8E	32
33	D8 BD 38 98 3A BA E8 C2 7F 76 26 33 95 2A 3D 80	48
49	5E 8A D7 BC 53 7C A3 E9 42 AB 87 4B 47 4E A8 7D	64
65	F1 77 9E 4C FE A1 5C 49 32 8F F4 A0 75 74 5A FB	80
81	AC FA C5 70 B9 97 C1 AA 40 C3 E0 73 5B D2 CC D0	96
97	3C 91 FD 6E AD 93 8C F5 62 C8 25 3B 34 39 4A BB	112
113	46 55 59 F7 B8 3F C0 88 BF 44 7B 2E B6 F9 D9 81	128

Basis 8 (XOR-Sequenzen):

01110101 11100101 01110100 00101100 10101010 10100001 11010011 00111101 ....

Bit Konversion: Basis 8 → Basis 7 (Indizes für Chiffre-Alphabet):

0111010 1111001 0101110 1000010 1100101 0101010 1000011 1010011 0011110 ....

(dez)	58	121	46	66	101	42	67	83	30 ....
(+1)	59	122	47	67	102	43	68	84	31....
Chiffre-Alphabet:									
(Text)	‡	D	=	ž	“	&	L	p	á ....
(hex)	87	44	3D	9E	93	26	4C	70	E1....

Nach allem ergibt sich der Chiffretext der Datei **Fontane.txt** wie folgt:

ASCII-Zeichensatz

‡D=ž“&Lpá<sup>a</sup>áîm/~ýÇ\*uYÙ<sub>3</sub>éú]ù»<sup>a</sup>Šè‡÷ñt\*LăøÂD2÷OI<4ØÀ.vp{n?½.□•ž’„;òg-\\Ø’Ù;G-  
òð}ú½)1ÚTÉ/Ú<sub>3</sub>òÓ)□q’çâ‡AÄx□Çw□êwDÚ9úšñOÚñž□ÂNHzoÉ0ÁÁ5<ç‡-,{<sup>a</sup>{á~UÉ\$}¿\³òî  
%o...F|Yða@<α-óĚÍZēnVúv...É?iYGâN»<sup>3</sup>ĩăóöèà1Úru5ûHμóMö’G=çwIàG|  
ÿĚă9^eÇEò1i?”„ò2u¾μÑŽ~ož—kNmJiv©~JÑiil5~mÁ!X+ÉĂo%=ó[g2á4Ăa—ĚÖD·ç□  
%omm#tŽçJoÀ~îâED‡4-’ôN¼v~hQoð~ùĂA~žCdÔ;€at|é’™cöCæv\_¶□°+ñ-  
5AĂaK†è’L{QèĂ~œVHØ¶™ö’0âgØi•tÂd¼cpNn|□...>p &ûTóŠp†œ<sup>a</sup>d/óĂ-  
TμXCOO×dÆWĂA9SöOĂĂœ!fĂă<.?ol¥d/A]ÍœœĂ»>Æi0ÆÁÇÖf%½¿37œLĂës{t’eo[4í&-  
=ôİáhvá1ox7ìp@„İi6öKİo•ă<ùio□\_ô%šo?μÉG{W„\_«]\_et{^àîñ

Hexadezimal:

87 44 3D 9E 93 26 4C 70 E1 AA E1 EE 6D 2F 7E FD C7 2A 75 59 D9 B8 E9 FA  
5D F9 BB AA 8A E8 87 F7 F1 74 2A 4C E3 F8 C2 44 32 F7 4F 49 3C 34 D8 C0  
2E 76 70 7B 6E 3F BD 2E 81 95 BF 9E 92 84 3B F2 67 96 5C D0 92 D9 3B 47  
2D F2 F0 7D FB BD 29 31 DA 54 C9 2F DA B8 F2 D3 29 8D 71 91 E7 E2 87 41  
C5 78 8F C7 77 81 EA 77 44 DA 39 FB 9A F1 4F DA F1 7A 8F C5 4E 48 87 6F  
CA 30 C1 C1 35 8B E7 87 2D B8 7B AA 7B E0 CD 7E 55 C9 A7 5D 5C BF 5C B3  
F2 CE 89 85 46 7C 59 F0 61 AE 3C A4 96 F3 CB CE 5A EB 6E 56 FA 76 85 C9

Als Folge der Bit-Konversion zeigt sich eine grundsätzliche Wirkung des Verfahrens:  
Die Länge des Chiffretextes verlängert sich im Verhältnis **7:8**. Aus 7 Klartext-Zeichen  
werden 8 Chiffretext-Zeichen. Somit entfallen auf jedes Klartext-Zeichen **8/7 = 1,143**  
Chiffretext-Zeichen. Die Forderung Klartext und Chiffretext sollten gleiche Länge haben,  
wird mithin nicht erfüllt [#2].

### e) Entschlüsselung

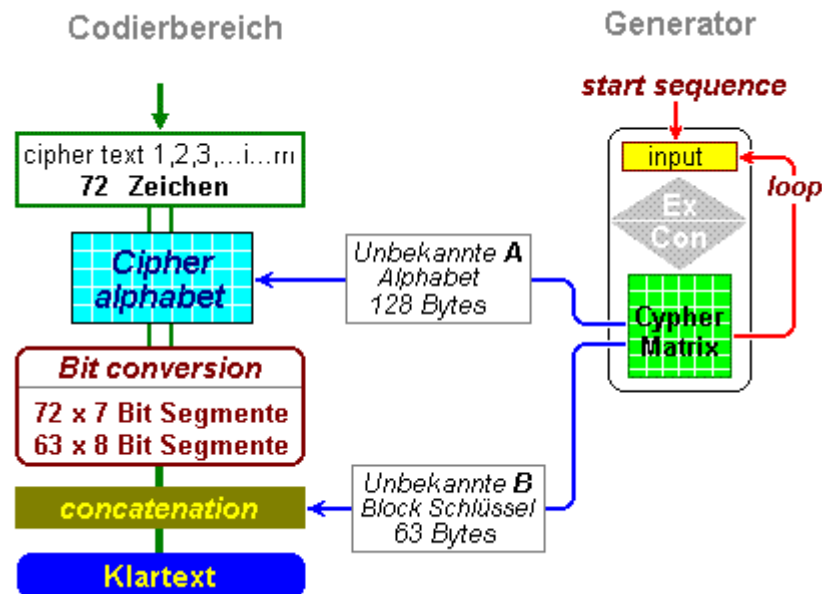
Für die Entschlüsselung erzeugt der Generator einen inhaltsgleichen Ablauf, wie bei der  
Verschlüsselung. Die Entschlüsselung wird im Codierbereich abgearbeitet, nur in der  
umgekehrten Reihenfolge:

1. Analyse 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 dezimalen Index-Werte der einzelnen Zeichen und verbindet deren

binäre Zahlen zu einer Bitfolge von **504** Bits. Diese Bitfolge wird wiederum in **63** 8-bit Sequenzen im Bitsystem zur Basis 8 aufgeteilt und mit dem entsprechenden **Block-Schlüssel** XOR-verknüpft. Als Ergebnis erscheint der ursprüngliche Klartext.

Die Entschlüsselung geschieht in jedem Durchgang wie folgt:



## f) Sicherheit des Verfahrens

Einem Angreifer sind grundsätzlich nur der **Chiffretext** und das **CypherMatrix** Verfahren bekannt. Das jeweilige Programm und die einzelnen Steuerparameter, 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.

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:

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

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

fx = funktionale Verbindung

an = 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 **an** zeigen sich somit als Gleichungen mit zwei unbekanntem 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 =  $k_1$  und dem in derselben Runde generierten Chiffre-Alphabet (128) =  $k_2$  gibt es keine Verbindung. Beide sind zwar aus der aktuellen CypherMatrix entnommen, haben aber keine funktionale Beziehung: ( $k_1 \rightarrow (Hk \text{ MOD } 169)+1$ ) und  $k_2 \rightarrow (Hk + Hp) \text{ MOD } 255+1$ ). 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" \[#3\]](#). Somit kann auch „brute force“ keinen Erfolg haben.

## 2. Hashberechnungen

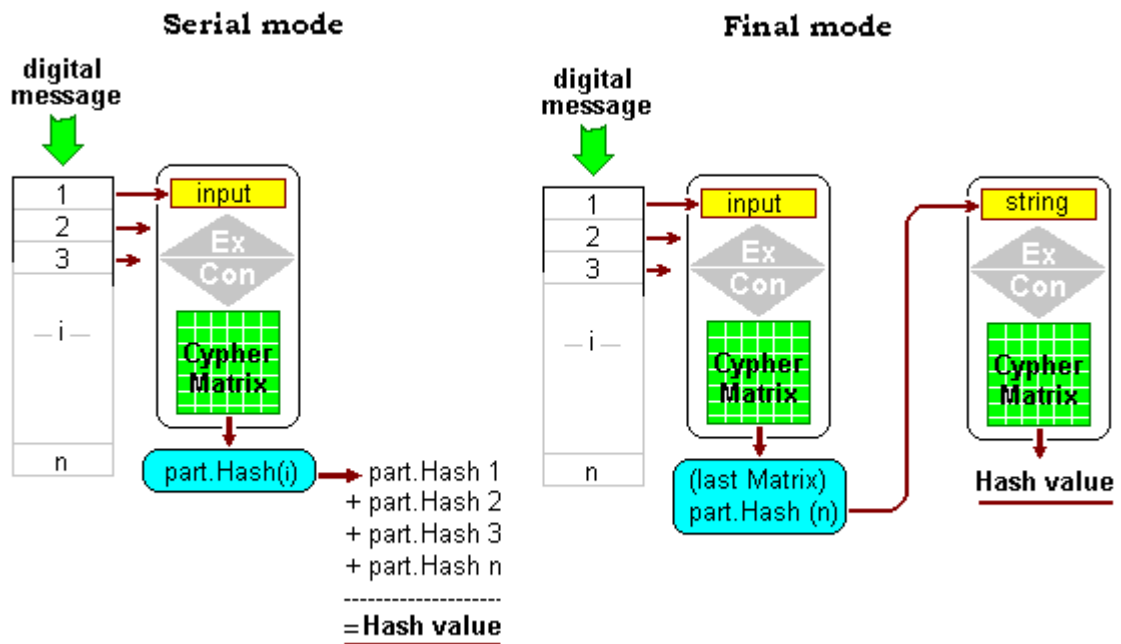
Das Ergebnis einer Hashfunktion kann ein einzelner **Wert**, eine eindeutige **Zeichenfolge** (Vektor) oder eine bestimmte **Struktur** (z.B: Matrix) sein. Im Cypher Matrix Verfahren werden digitale Sequenzen (Dateien) in Textblöcke aufgeteilt (z.B. 64 Bytes), die in jeder Runde positionsgewichtet, mit der Hashkonstante **C(k)** multipliziert, zu einer Hashfunktionsfolge erweitert, wieder zur BASIS VARIATION verdichtet und abschließend einer dreifachen Permutation unterworfen werden. Ein Schlüssel ist nicht erforderlich. Der erste Textblock wird als Startsequenz verwendet. Die letzte Cypher Matrix mit 16x16 Elementen ergibt den **Hashwert (H)**.

Nach den Gesetzen der Wahrscheinlichkeit tritt eine Wiederholung identischer Strukturen erst in **256!** (Fakultät) = **8E+506** Fällen auf. .

### a) Alternative Hashwertberechnungen

Zwei Methoden der Hashwertberechnung sind möglich:

- a) Serielle Berechnung von partiellen Hashwerten in jeder Runde und Addition aller Rundenwerte zum Hashwert (**serial mode**) und
- b) Berechnung des Hashwertes aus der CypherMatrix der letzten Runde unter Berücksichtigung aller vorhergehenden partiellen Hashwerte (**final mode**).



Die Struktur der CypherMatrix als Hashwert (Term) ist allerdings recht unhandlich. Das Ergebnis wird daher in prägnante Größen umgeformt. Die Elemente der CypherMatrix werden seriell in einem CypherString mit der Länge  $n = 256$  geordnet, der abschließend der Hashfunktion als zusätzliche Eingabesequenz übergeben wird. Der Hashwert wird berechnet und das Ergebnis als Zahl im Zahlensystem zur **Basis 62** ausgewiesen.

Das Zahlensystem zur Basis 62 umfasst die folgenden Ziffern:

0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ  
 abcdefghijklmnopqrstuvwxyz  
 (definiert vom Autor, nicht standardisiert)

Als Beispiel Hashwertberechnungen für die Datei **Fontane.txt** (432 Bytes):

1. Berechnung im „serial mode“ [CMH-3 sm]

**CMH-3 sm** / 64 / 62 / 77 / 1

Serielle Matrixwerte in Fontane.txt

dezimal	Basis 62	Runde	CM(k)
8943474207281465	exasTwXSj	1	2L2vwb
9172081797999652	g0Vdtxnam	2	2L3WB4
9156559301634833	fw6MOywuv	3	2L3cq1
8709176174651337	dt3vgqveL	4	2L2tBT
9186069556078276	g4TuALCku	5	2L3xUg
8847891854268068	eWS6445bw	6	2L3R4U
8988221739082929	fAlgPrI3J	7	2L3qbx

**Serieller Hashwert** für: Fontane.txt

dezimal: 63003474630996560

Basis 16: DFD5538CFEF250

Basis 62: **4eYVV81H2u**

=====

2. Berechnung im „**final mode**“ [CMH-3 lcx]  
 Cyphermatrix mit dreifacher Permutation (Version: B)

**CMH-3 lcx** / 64 / 62 / 77 / 1  
 Serielle Matrixwerte in Fontane.txt

dezimal	Basis 62	Runde	CM(k)
9378932591334404	gxFMnqTNE	1	2L3mgw
8900817014962569	eITsFMlln	2	2L2zJl
8556594378776745	dBjdpT2td	3	2L2S9l
9214891372794113	gCfKT8uif	4	2L3sRb
9123485721452161	fmi56n1wv	5	2L3puh
8806090838808033	eKaANv9xB	6	2L3M0X
8914931202673337	epUMVy1l9	7	2L3hjX

dezimal: 62895743120801362  
 Basis 16: DF73585A997E52  
 Basis 62: 4e3upMVPUY

Matrixwert letzte Runde:  
 dezimal: 8913973467573561  
 Basis 16: 1FAB362E126D39  
 Basis 62: epDV6V8zB

**Finaler Hashwert** für: FONTANE.TXT  
 dezimal: 71809716588374923  
 Basis 16: FF1E8E88ABEB8B  
 Basis 62: **5lt8KT0YTj**

3. Berechnung im „**final mode**“ [CMH-3 lc]  
 Cyphermatrix ohne Permutation (Version: 0)

**CMH-3 lc** / 64 / 62 / 77 / 1  
 Serielle Matrixwerte in Fontane.txt

dezimal	Basis 62	Runde	CM(k)
8943474207281465	exasTwXSj	1	2L2vwb
9172081797999652	g0Vdtxnam	2	2L3WB4
9156559301634833	fw6MOywuv	3	2L3cq1
8709176174651337	dt3vgqveL	4	2L2tBT
9186069556078276	g4TuALCku	5	2L3xUg
8847891854268068	eWS6445bw	6	2L3R4U
8988221739082929	fAlgPrI3J	7	2L3qbx

dezimal: 63003474630996560  
 Basis 16: DFD5538CFEF250  
 Basis 62: 4eYVV81H2u

Matrixwert letzte Runde:  
 decimal: 8987256145720881  
 Basis 16: 1FEDDCA2123631  
 Basis 62: fA1gQaPkf

```

-----
Finaler Hashwert für: FONTANE.TXT
dezimal: 71990730776717441
Basis 16: FFC3302F112881
Basis 62: 5JiXBYbgnZ
=====

```

Der Wert **CM(k)** ist die Positionsgewichtung zur Vermeidung von Kollisionen. Die Länge der Hashwerte im Programm **CMH-3 \*\*\*.exe** bewegen sich zwischen **9** bis **12** Ziffern im Zahlensystem zur Basis 62 (Spanne: **62<sup>9</sup>** bis **62<sup>12</sup>**). Für die Datei **Fontane.txt** berechnen aktuelle Hashwertfunktionen die folgenden Hashwerte:

**MD 5**

Basis 16: EB 14 95 AE 71 A8 2A 55 9F 14 77 09 61 5B CB 28

**SHA 1**

Basis 16: FA 7E CE C4 31 55 4D 26 28 E7 82 E2 56 07 86 49 9D DB 3E

**RIPMD-160**

Basis 16: 2F C3 29 05 A1 27 AA 34 D7 FF 19 DB D3 19 60 27 2E 35 8E 4E

**CMH-3 lcx**

Basis 62: 5lt8KT0YTj

Basis 16: FF1E8E88ABEB8B

Basis 10: 71809716588374923

**2. Sensibilität der Hashfunktion**

Zur Darstellung der Sensibilität wird in der Datei **Fontane.txt** die letzte Ziffer der letzten Zeile von **..98** auf **..99** geändert:

**(Fontane.txt)**

..... Theodor Fontane, 1819-1898

..... 00110001 00111001 00101101 00110001 00111000 00111001 00111000

**(Fontane1.txt)**

..... Theodor Fontane, 1819-1899

..... 00110001 00111001 00101101 00110001 00111000 00111001 00111001

char	=	bit
8	=	1000
9	=	1001

Das letzte Bit „0“ wird auf „1“ gesetzt. Im Übrigen bleibt alles unverändert. Die Berechnung des Hashwerts mit dem geänderten Bit führt zu folgenden Ergebnissen:

```

Summe Runden 1 bis 7:
dezimal: 62836912216429994
Basis 16: DF3DD6B77155AA
Basis 62: 4dnD4nQ7je
-----

```

Matrixwert letzte Runde:  
dezimal: 8855148871273361  
Basis 16: 1F75B602E7CB91  
Basis 62: eYVrQJshd

-----  
**Finaler Hashwert** für: FONTANE1.TXT  
dezimal: 71692061087703355  
Basis 16: FEB38CBA59213B  
Basis 62: **5ILiWdk0RH**  
=====

Der durch Änderung auch nur des letzten Bits verursachte Unterschied in den Hashwerten beträgt:

	Original	Änderung	Differenz
Basis 62:	<b>5It8KT0YTj</b>	<b>5ILiWdk0RH</b>	<b>XP0FGY2S</b>
dezimal:	71809716588374923	71692061087703355	11765550071568

Der grundsätzliche Unterschied zwischen aktuellen Hashfunktionen und den CypherMatrix Hashwertberechnungen zeigt sich bereits in den Grundelementen. Die folgende Tabelle zeigt eine Übersicht der grundsätzlichen Unterschiede:

<b>Hashfunktionen</b> (allgem.)	<b>CypherMatrix Hashfunktion</b>
<b>Grundelemente</b>	
<b>Bits</b>	<b>Bytes</b>
<b>Zusatzfunktionen</b>	
IVs, SALT, padding	keine
<b>Arbeitsschritte, Sequenzen</b> (message digest)	
224, 256, 384, 512, 1024 bits (teilweise: variabel)	kontinuierlich, unbegrenzt (optimal: 16 bis 256 Bytes)
<b>Interne Beschaffenheit</b> (internal states)	
Feistel network keys, S-boxes, constants, shifting, rotation, mixing, XOR swapping, permutations	positionsgewichtet Hash-Konstante C Expansion, Kontraktion keine Kollisionen
<b>Kompressionsfunktion</b>	
„Merkle-Damgård“ block ciphers, AES-based Threefish based	keine
<b>Ausgabefunktion</b>	
output function truncated to output fixed length	CypherMatrix: $GF(16^2)$ dreifache Permutation Zahlensystem zur Basis 62 (9 bis 11 Ziffern)
<b>Anwendungen</b> (applications)	
hashing, MAC, randomizing, PRNG, digital signatures, authentication, encryptions,	Hashfunktion, Zufallsfolgen digitale Signaturen, RNG, Authentifizierung, Verschlüsselungen



Matrixwert letzte Runde:  
dezimal: 8724051910203152  
Basis 16: 1EFE7A9F1C1710  
Basis 62: dxHpDnJo0

-----  
**Finaler Hashwert** für: CMH-3lcx.EXE  
dezimal: 2.5014875292881749E+19  
Basis 16: 15B26B08B5CE98414  
Basis 62: **TnsORQ3irOW**

=====

Die Hashwerte im CypherMatrix Verfahren sind **Zahlen** in den betreffenden Zahlensystemen, keine **Zeichen** wie in herkömmlichen Hashwerten.

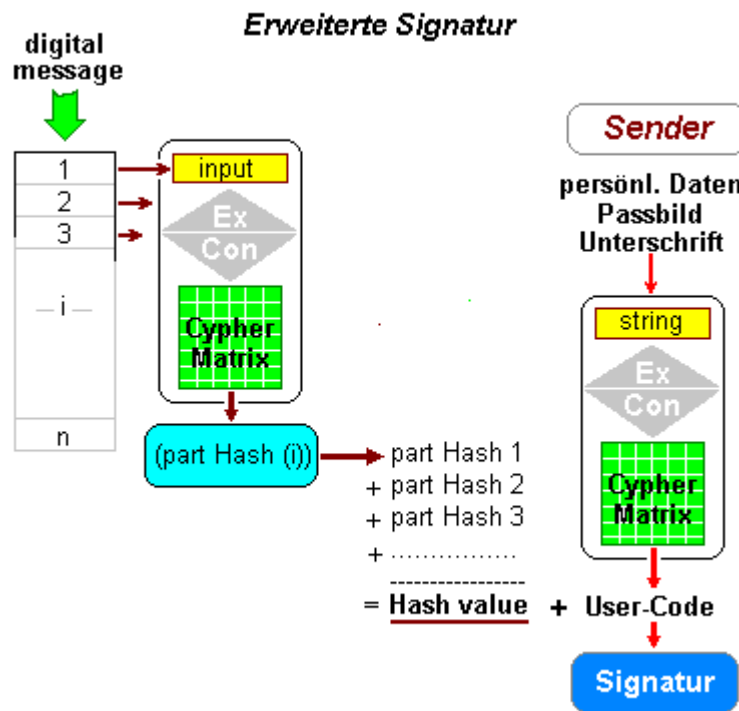
Abschließend zeigt sich, dass zwischen derzeitigen Hashfunktionen und der CypherMatrix Hashfunktion keinerlei Bezugspunkte vorhanden sind. Die hier beschriebene Hashfunktion geht völlig neue Wege, ohne Kollisionen, aber mit großer Sensibilität („Avalanche effect“).

### 3. Digitale Signaturen

Der Informationsaustausch über digitale Netze (Intranet, Internet) ist grundsätzlich unsicher (Integritätsproblem, Authentizitätsproblem). Eine Lösung bietet hier die elektronische Signatur. Entsprechend der datentechnischen Gestaltung werden unterschieden:

- a) einfache elektrische Signatur,
- b) fortgeschrittene elektronische Signatur und
- c) qualifizierte elektronische Signatur.

Für qualifizierte elektronische Signaturen schreibt das Signaturgesetz vom 16.05.2001 [SigG, BGBl.I S.876] feste Verfahrensschritte und Schlüsseltechniken vor. In rechtlich weniger bedeutsamen Fällen - aber dennoch mit **vertretbarer Sicherheit** - bieten sich digitale Signaturen auf Basis des CypherMatrix Verfahrens an.



Wird die CypherMatrix Hashfunktion sowohl auf die zu signierende Nachricht als auch auf die persönlichen Daten des Signierenden (ID) (einschließlich persönlichem Foto und Unterschrift) angewendet, kann eine erweiterte digitale Signatur begründet werden. Zum Signieren einer digitalen Information (Nachricht, Datei, e-mail) werden der Hashwert der **persönlichen Daten** des Senders (Signatar) und der Hashwert der zu **signierenden Information** addiert und zu einer **digitalen Signatur** zusammengefasst.

Die Einzelheiten sind in den folgenden Artikeln zusammengefasst:

[Erweiterte digitale Signatur](#) [#4]

[>teleCode< als erweiterte Signatur](#) [#5]

[Alternativer elektronischer Personalausweis](#) [#6]

Die **digitale Signatur** enthält eine eindeutige Information über die Nachricht und den Signatar in verkürzter Form. Wird die digitale Signatur zusammen mit der entsprechenden Nachricht übermittelt, kann der Empfänger

- a) den **Hashwert** der Nachricht errechnen und
- b) aus der Differenz zur erhaltenen „digitalen Signatur“ die **Identität** des Absenders ermitteln.

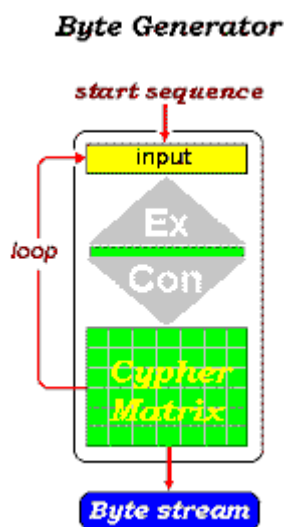
Sind die ID-Daten des Absenders in einer Internet-Datenbank (Signatur Portal) gespeichert – selbstverständlich gesichert und verschlüsselt –, kann der Empfänger persönlich die Integrität der Nachricht und die Identität des Absenders testen.

Die in den vorstehenden Artikeln erläuterten Verfahren sind eigene Entwicklungen und haben mit den traditionellen Signaturprogrammen wenig Gemeinsamkeiten.

Der einzige Berührungspunkt ist die Tatsache, dass neben den Beteiligten eine dritte außenstehende Instanz (Zertifizierungsstelle, Internet-Datenbank) vorhanden sein muss, um die Vertraulichkeit (Identität des Signatars) und Sicherheit (Integrität der Nachricht) zu gewährleisten.

#### 4. Unbegrenzte Byte- und Zahlenfolgen

Mit dem Generator des CypherMatrix Verfahrens können unbegrenzte **Bytefolgen** (byte stream) erzeugt werden. Dabei ist nur die Startsequenz als Initialisierungsfaktor und die Länge der gewünschten Folge einzugeben. In jedem Durchlauf der Basisfunktion werden die 256 Elemente der letzten CypherMatrix seriell, entsprechend ihrer Verteilung, in aufeinander folgenden Zeilen in einer Ergebnisdatei gespeichert. Als quasi "**random series**" sind sie für vielfältige kryptographische Aufgaben einsetzbar (z.B. als Schlüsseldateien für "one-time-pad" und "multi-time-pad" Verschlüsselungen).



Wenn andererseits in der Basisfunktion die **Kontraktion** alternativ mit **MODULO 8, 16, 32, 64** oder andere) anstelle **MODULO 256** vorgenommen wird (Kontraktion zur BASIS VARIATION), können weitere unbegrenzte Folgen generiert werden.

Ein Beispiel mit der Startsequenz „**Bruno der Braunbär aus Bregenz im Breisgau**“ für eine Kontraktion mit **MODULO 16** und **23** Zeilen:

```

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

```

10 16 12 4 11 13 1 8 5 14 6 2 15 3 7 9
5 1 13 4 14 15 7 16 11 6 2 12 9 3 8 10
7 10 11 14 12 1 5 6 13 15 8 16 9 2 3 4
13 2 3 1 4 14 5 15 8 11 9 7 16 10 12 6
1 9 15 11 10 12 13 2 3 16 7 4 6 8 5 14
11 15 8 5 1 4 7 16 2 13 3 6 12 9 10 14
6 8 13 10 12 2 11 14 3 5 7 1 15 16 9 4
16 4 5 6 13 7 10 2 3 11 12 1 8 9 14 15
5 2 9 7 13 15 14 16 6 4 3 1 8 10 11 12
13 16 14 7 10 9 4 5 3 6 8 15 11 12 1 2
15 13 8 9 7 10 14 4 16 11 12 5 1 2 3 6
15 4 14 13 16 1 2 10 3 7 5 6 8 9 11 12
.....

```

Die erzeugten Zahlenfolgen – hier **1** bis **16** im Umfang von 23 Zeilen – können unbegrenzt verlängert und erweitert werden. Eine Wiederholung der gleichen Zeile tritt nach den Grundsätzen der Wahrscheinlichkeit erst in **16!** (Fakultät) = **20 922 789 888 000** Fällen ein.

#### D. Abschließende Bemerkungen

Auf der Grundlage einer sinnvollen Systematisierung der Bitfolgen, angelehnt an die Systematisierung in der Zahlentheorie, reichen Umwandlungen von einem Bitsystem in ein anderes Bitsystem für eine einfache und sichere Verschlüsselung aus: z.B. Umwandlungen vom Bitsystem zur Basis 8 in das Bitsystem zur Basis 7 oder umgekehrt von Basis 7 nach Basis 8. Auch Konversionen von Basis 8 nach Basis 9 oder Basis 10 und zurück nach Basis 7 bringen sichere Verschlüsselungen [#7]. Die dazu erforderlichen Zeichen-Alphabete sind allerdings beachtlich umfangreicher (Zahlensysteme zur Basis 32 und 64) [#8]. Sogar die XOR-Verknüpfung kann entfallen, wenn die Bitfolgen des Klartextes im Bitsystem zur Basis 8 direkt zu Bitfolgen im Bitsystem zur Basis 7 umgewandelt werden und als Indizes für das Chiffre-Alphabet mit 128 Zeichen dienen.

Weitere Einzelheiten zum **CypherMatrix** Verfahren unter: [www.telecypher.net/](http://www.telecypher.net/)

München, im **April 2011**

**Kryptologie mit CRYPTOOL:** [www.cryptool.org](http://www.cryptool.org)

- [#1] [telecypher.net/Kollfrei.pdf](http://telecypher.net/Kollfrei.pdf)
  - [#2] [telecypher.net/EQUILANG.HTM](http://telecypher.net/EQUILANG.HTM)
  - [#3] [telecypher.net/CYPHKERN.HTM#Z32](http://telecypher.net/CYPHKERN.HTM#Z32)
  - [#4] [telecypher.net/Signatur.pdf](http://telecypher.net/Signatur.pdf)
  - [#5] [telecypher.net/teleCode.pdf](http://telecypher.net/teleCode.pdf)
  - [#6] [telecypher.net/Personalausweis.pdf](http://telecypher.net/Personalausweis.pdf)
  - [#7] [telecypher.net/CYPHKERN.HTM#Z45](http://telecypher.net/CYPHKERN.HTM#Z45)
  - [#8] [telecypher.net/DEnumsys.pdf](http://telecypher.net/DEnumsys.pdf)
-