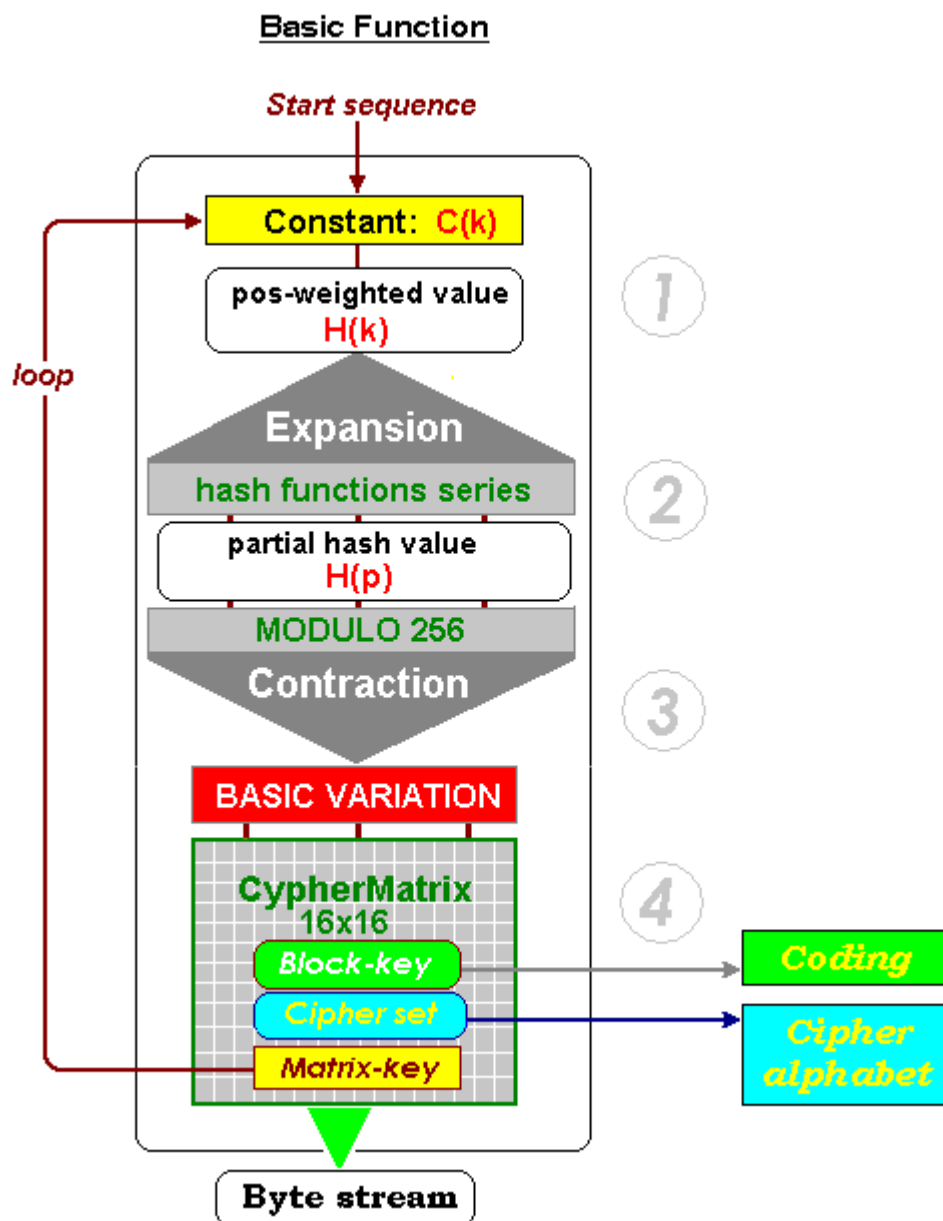


# Kryptographische Basisfunktion in Byte-Technik

(Ernst Erich Schnoor)

## I. Basis Funktion

Zuerst wird eine grundlegende Funktion vorgestellt, die mit der Eingabe einer kurzen Information (**Start Sequenz**) beginnt und diese Information zu einem bestimmten Hash-Ergebnis entwickelt (vom Autor „**CypherMatrix**“ Verfahren genannt: DPMA 19811593 vom 18.03.1998) Die Funktion verwendet nur **Bytes**, einfache Mathematik, MODULO Rechnungen eingeschlossen, und Zahlensysteme von zur Basis 2 bis zur Basis 256, aber vor allem keine Manipulationen mit **Bits**. Die folgende Skizze veranschaulicht die Zusammenhänge:



Dann wird in praktischen Beispielen gezeigt, wie die Basisfunktion mit anderen Prozessen verbunden wird, um kryptographische Probleme zu lösen (beispielsweise: Verschlüsselungen, Hashberechnungen, Signaturen und Zufallsfolgen).

Die Funktion beginnt mit der Eingabe einer **Startsequenz** (z.B.: 42 Bytes). Die Startsequenz steuert den gesamten Verlauf. Die Änderung auch nur eines einzigen Bits führt zu völlig unterschiedlichen Bestimmungsfaktoren.

Als Beispiel 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 von bestimmten Bytes **a(i)** mit der Länge (**n**): z.B. Nachricht, Passphrase, Blockschlüssel, Pixel, Klänge usw. Um die Folge als Sachverhalt zu analysieren, wird sie mit dem Wert **H(k)** gekennzeichnet. Dazu wird jedem Byte **a(i)** ein Index zugeordnet und alle (**n**) Bytes werden in sachgerechter Weise miteinander verknüpft.

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

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

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

$$H(k) = 3993$$

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



## 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 – eindeutig bestimmt wird durch seine Koordinaten für **Gegenstand**, **Ort** und **Zeit** (kartesisches Koordinatensystem).

Wir definieren:

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

Die Zeit ist nur dann von Bedeutung, wenn eine variable Beziehung zwischen einzelnen Bytes **a(i)** und der Prozessfrequenz besteht, vielleicht im Fall „voice streaming“. Normalerweise ist die Verbindung jedoch konstant und wir können **t = 1** setzen.

Um eine eindeutige Bestimmung für jedes Byte **a(i)** zu erhalten, verknüpfen wir **Gegenstand**, **Ort** und **Zeit** durch Multiplikation ihrer Dimensionswerte und addieren jedes Ergebnis zum Bestimmungswert **H(k)**.

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

$$H(k) = 86560$$

Jedes Byte **a(i)** wird mit seiner Position **p(i)** multipliziert, d.h. **positionsgewichtet**. Damit unterscheiden sie sich grundsätzlich. Aber Kollisionen als Folge eines Austausches von Bytes innerhalb der Information sind noch nicht ausgeschlossen. Um das zu erreichen erhöhen wir den Wert eines Bytes **a(i)** in einen Bereich oberhalb der Länge (**n**) der Zeichenfolge. Wir erreichen dies durch Erweiterung von **p(i)** mit der Konstanten **C(k)**. Diese Konstante hat die Aufgabe, Kollisionen zu vermeiden: "Bestimmungsfaktoren für Kollisionsfreiheit" erläutern die Einzelheiten.

$$C(k) = n * (n - 2) + \text{code}$$

$$C(k) = 1681$$

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

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.

2

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

Um eine größere und prüfbare Sicherheit zu erreichen, wird eine „**Expansionsfunktion**“ eingeführt, die die Bestimmungsfaktoren auf möglichst viele Variablen erweitert (etwa 160 bis 2400 Zeichen). Hierzu wird der Wert eines jeden Zeichens **a(i)**

mit der Position  $p(i)$  und dem Zwischenwert  $H(k)$  verknüpft und in einen höheren Bereich hochgerechnet (multipliziert). Das jeweilige Ergebnis  $s(i)$  rechnet das Programm um in  $d(i)$ , einen Wert in einem höheren Zahlensystem (z.B. zur **Basis 77**). Die Ziffern  $d(i)$  werden in serieller Reihenfolge in einer **Hash Funktionsfolge** gespeichert. Gleichzeitig errechnet das Programm einen zusätzlichen Bestimmungswert  $H(p)$  als Summe aller Werte  $s(i)$  in jeder Runde  $r$  ( $r_1, r_2, \dots, r_j, \dots, r_m$ ).

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

$$S_i \rightarrow d_i \quad (\text{Basis 77})$$

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

$$H(p) = 588503523067$$

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

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

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

Zchn	$p_i$	$H_k$	$(a_i+1)*p_i*H_k$	$S_i$	Basis 77			
$(a_i+1)$	$(a_i+1)*p_i$		$p_i+code+r$					
..	...	..	.....	.....	.....			
<b>B</b>	67	11	737	6798793	5010710441	13	5010710454	<b>1àfhêv</b>
<b>r</b>	115	12	1380	6798793	9382334340	14	9382334354	<b>3ZäL28</b>
<b>a</b>	98	13	1274	6798793	8661662282	15	8661662297	<b>3FUrvp</b>
<b>u</b>	118	14	1652	6798793	11231606036	16	11231606052	<b>4Bcè#p</b>
<b>n</b>	111	15	1665	6798793	11319990345	17	11319990362	<b>4E1gçà</b>
<b>b</b>	99	16	1584	6798793	10769288112	18	10769288130	<b>3êRM9&amp;</b>
<b>ä</b>	133	17	2261	6798793	15372070973	19	15372070992	<b>5qMP1I</b>
<b>r</b>	115	18	2070	6798793	14073501510	20	14073501530	<b>5FQâ3B</b>
..	...	..	.....	.....	.....	..	.....	.....
				Summe H(p)	588503523067			<b>2#WECDΩ</b>

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

CèxèâibGQ3ãZàOp18âBYà1VNn#ScMoãq1xuzçN23#4Uu2kDZ1j##XbY**1àfhêv3ZäL283FU**  
**rpv4Bcè#p4E1gçà3êRM9&5qMP1I5FQâ3B**1iKW#W4æ5wâv6HJçlb6VhÉBZ1ázFWE42ërÉH7  
 H3Btr6oâEM&746Stà7DSFUb86i62G9KpOWQ2hàvkt8e2XTc994yPk2#06435âfKççAUr#6  
 K9aâKQeA92äd@BRéXYMAYhp8iA77écvCYdâaç

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>EπS1</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	Element - Theta	
<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). In jeder Runde generiert das Verfahren aus allen Elementen der BASIS VARIATION die „**CypherMatrix**“ mit 16x16 Elementen entweder mit drei Permutationen oder direkt in einem Vorgang.



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

### CypherMatrix (16x16) abgeleitet aus der BASIS VARIATION (256 Elemente)

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

Die **CypherMatrix** begründet in ihrer Struktur (**GF 16<sup>2</sup>**) eine eindeutige und kollisionsfreie Form als "**Hashwert**". Nach den Grundsätzen der Wahrscheinlichkeit wird ein gleicher Wert erst in **256!** (Fakultät) = **8.578E+505** Fällen auftreten.

Die folgenden Parameter zur Steuerung von Verschlüsselungen holt das Programm aus der laufenden CypherMatrix:

**Alpha:**  $((H(k) + H(p) \text{ MOD } 255) + 1) = 36$ , legt den Beginn des **Chiffre-Alphabets** von 128 Elementen fest:

33			BC	9E				8C	F7	D4	A5	C6	71	BE	85	E7	48
49		BA	53		75	E0	F5	B8		66	B5	E6	9F	6F		35	64
65	E8	7C		74		62	3F	F6	F0	68	56	7A	8B		50	CF	80
81	A3	4C	5A	73	C8	C0	28	79	EB	6A	6B	9B	EC	F2	A4	C2	96
97	FE	FB	5B	25	88	86	C9	89	30		6C	6D	AF	CE	7F	E9	112
113	AC	D2	3B	BF	63	C4		EA	52	60	9A	4F	AE	76	42		128
129	CC	34	44	78		45	2D	72	54		58	3E	26	AB	A1	FA	144
145		7B	E5		CA	2B	69	9C	2F	99		33	87	5C	C5	D0	160
161		FC	A7	64	E2	D3	ED			E1	95	4B					176
177	82	43	8D	5F	41	51	D6	5D	8E	2A			70				192

Bestimmte Zeichen (Hex: **00 bis 20, 22, 2C** und weitere) werden ausgeklammert, weil sie in einigen Situationen noch ihre ursprünglichen Aufgaben wahrnehmen (z.B. **1A** =ASCII-26) und die ordnungsgemäße Durchführung des Programms stören.

**Beta:**  $(H(k) \text{ MOD } 169) + 1 = 93$ , bestimmt den Offset, ab dem 63 Bytes für den laufenden **Block-Schlüssel** entnommen werden:

81													EC	F2	A4	C2	96
97	FE	FB	5B	25	88	86	C9	89	30	02	6C	6D	AF	CE	7F	E9	112
113	AC	D2	3B	BF	63	C4	16	EA	52	60	9A	4F	AE	76	42	18	128
129	CC	34	44	78	DE	45	2D	72	54	22	58	3E	26	AB	A1	FA	144
145	03	7B	E5	B2	CA	2B	69	9C	2F	99	DF						160

Die Länge der Klartext-Blocks und ebenso die Länge der Block-Schlüssel sind grundsätzlich gleich. Sie können alternativ mit 35, 42, 49, 56, **63** oder 70 Bytes (Vielfaches von 7) gewählt werden. Block-Schlüssel sind nur in Verschlüsselungen mit XOR-Verknüpfungen erforderlich.

**Gamma:**  $((H(p) + \text{code}) \text{ MOD } 196) + 1 = 49$ , bestimmt den Offset für die Entnahme des **Matrix-Schlüssels**:

49	0D	BA	53	1F	75	E0	F5	B8	1C	66	B5	E6	9F	6F	0F	35	64
65	E8	7C	14	74	0C	62	3F	F6	F0	68	56	7A	8B	DB	50	CF	80
81	A3	4C	5A	73	C8	C0	28	79	EB	6A							96

Zur Initialisierung des nächsten Durchlaufs wird der Matrix-Schlüssel auf den Beginn der Runde zurückgeführt (**loop**). Die Folge der Matrix-Schlüssel steuert den gesamten Ablauf des Verfahrens inhaltsgleich, sowohl beim Sender als auch beim Empfänger.

Die Sensibilität der Basisfunktion wird im Anhang ["Wechsel in der Start sequenz"](#) dargestellt.

## II. Codier Maschine

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

„**CypherMatrix**“ entwickelt sich so zu einem allgemeinen kryptographischen Werkzeug, das in fast allen Gebieten der Kryptographie eingesetzt werden kann, sozusagen als „**Codier Maschine**“. Sie liefert alle Steuerungsparameter für kryptographische Anwendungen (z.B. Matrix-Schlüssel, Chiffre-Alphabete, Block-Schlüssel, Bytefolgen, Hashwerte, Authentifizierungs Merkmale, S-Boxen usw.). Die **Matrix-Schlüssel** allein steuern in jeder Runde das gesamte Verfahren.

Die **Basis Funktion** kann auch in erweiterter Form mit **10-Bit** Sequenzen im Bytesystem zur **Basis 10** als CypherMatrix  $GF(32^2)$  [ $32 \times 32 = 1024$  Elemente] oder mit **12-Bit** Sequenzen im Bytesystem zur **Basis 12** als CypherMatrix  $GF(64^2)$  [ $64 \times 64 = 4096$  Elemente] durchgeführt werden.

Mit der **Basis Funktion** als Grundlage sind insbesondere folgende Anwendungen in Byte-Technik entwickelt worden:

Durchführung von Verschlüsselungen mit verschiedenen Operationen,  
Berechnung von Hashwerten (CypherMatrix Hash),  
einfache und erweiterte Signaturen (telePortal) und  
Generierung unbegrenzter Bytes- und Zahlenfolgen.

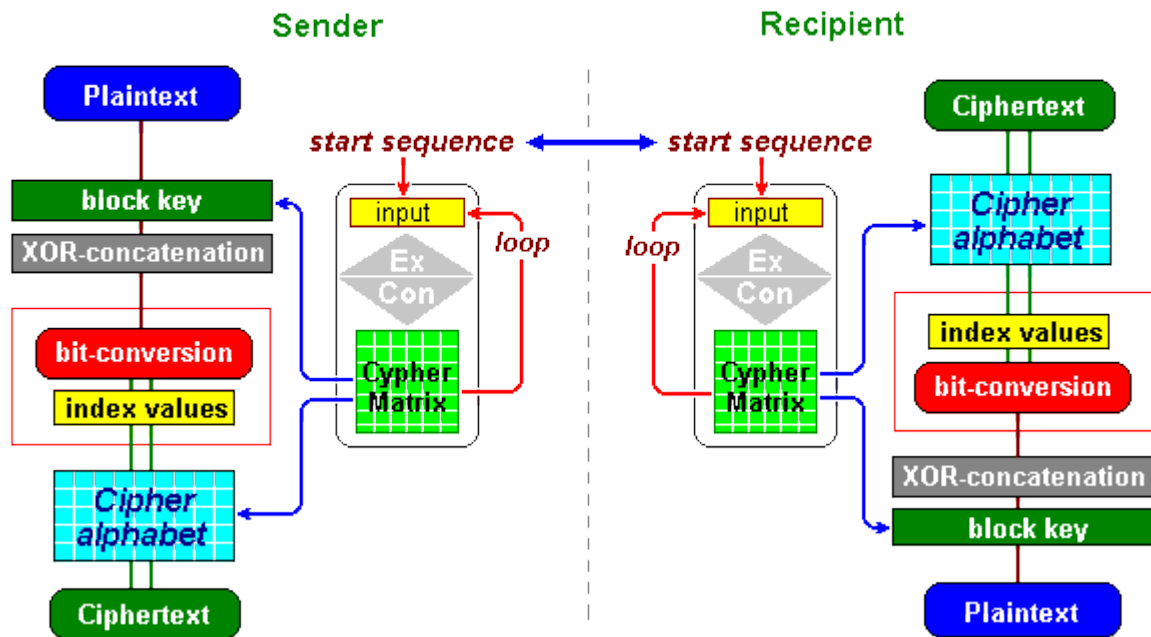
### A. Verschlüsselungen

Zur Durchführung von Verschlüsselungen wird die **Basis Funktion** mit geeigneten Zusatzfunktionen – **Codierbereiche** – kombiniert. Dabei stellt die jeweilige **CypherMatrix** in jedem Durchlauf die zur Verschlüsselung notwendigen Steuerungsparameter zur Verfügung. Für jedes Programm sind folgende Parameter erforderlich:

Länge des >Matrix-Schlüssels<:	36 – 64 Bytes
Länge des Block-Schlüssels<:	35 – 91 Bytes
Zahlensystem für Expansionsfunktion:	35 – 96 Grundziffern
individueller Anwender-Code:	1 - 99 Zahl

Das folgende Schema zeigt die Durchführung von Verschlüsselungen in der Kombination von **Basis Funktion** und **Codierbereich**:

## Encryption / Decryption scheme



Als symmetrisches Verfahren geben **Sender** und **Empfänger** eine identische **Startsequenz** ein, die das gesamte Verfahren steuert. Diese Passphrase muss einprägsam sein, etwas ausgefallen und möglichst mit spaßigen Worten die leicht im Gedächtnis bleiben und darf nicht irgendwo niedergeschrieben werden. Einige Beispiele:

Baron Münchhausen reitet über den Bodensee	[42 Bytes]
Auf der Fischbachalm gibt es keine Heringe	[42 Bytes]
Sven Hedin is sailing around the Northpole	[42 bytes]

Startsequenzen sollten etwa **42 Bytes** lang sein, damit wegen ihrer Länge Lexikon-Angriffe und iteratives Suchen unmöglich sind. Ein Angreifer ist auch nicht in der Lage, Teile der Startsequenz weder einzeln noch nacheinander zu analysieren, weil die Startsequenz nur als Ganzes gefunden werden könnte, wenn überhaupt.

Um die Verschlüsselung sicherer zu machen wird eine „**Bit Konversion**“ eingeführt, ein Schritt, der die Ergebnisse der XOR-Verknüpfung von 8-Bit-Segmente in 7-Bit-Segmente (Werte von 0 bis 127) umgruppiert. Die Reihenfolge der Bits „0“ und „1“ bleiben unverändert. Es erfolgt nur eine neue Segmentierung (8-Bit → 7-Bit: Wechsel im Bytesystem, Bit-conversion).

8-Bit XOR-Segmente umgruppiert in 7-Bit-Segmente als "**Indexwerte**" für das Chiffre-Alphabet von 128 Zeichen:

8-Bit: **11101111100001111001101010101111101010011101111101000100** ....  
 7-Bit: **11101111100001111001101010101111101010011101111101000100** ....

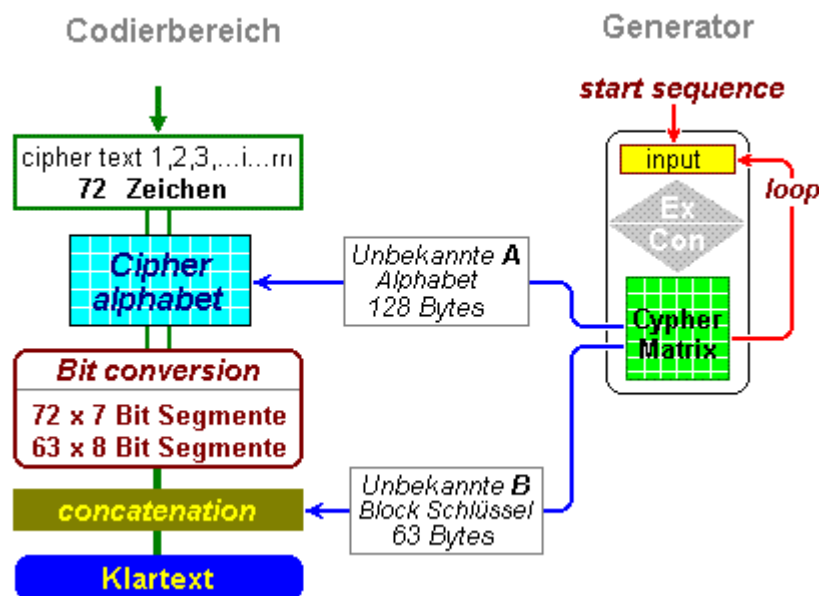
Die umgruppierten 7-Bit Segmente sind nur **Indexwerte** (Zeiger) auf Positionen

im Array **Chiffre-Alphabet**. Das Verfahren ist dem „Coding base 64“ ähnlich, in dem 8-Bit Segmente in 6-Bit Segmente aufgeteilt werden, die zugeordnete Zeichen aus einem 64-Zeichen Alphabet ansprechen.

Das Verschlüsselungsverfahren wird als „**dynamisches one-time-pad**“ durchgeführt. In jedem Durchlauf wird ein **Block-Schlüssel** von **63 Bytes** Länge (aus der laufenden CypherMatrix) mit dem entsprechenden **Klartext-Block** gleicher Länge XOR-verknüpft und kein Block-Schlüssel wird wiederholt in den weiteren Runden.

Die Sicherheit des Verfahrens kann anhand folgender Tatsachen beurteilt werden: Grundsätzlich kennt ein Angreifer nur den **Chiffre-Text** und die **Länge** der verschlüsselten Nachricht. Weiter möge ihm die „CypherMatrix“ Methode mit ihren drei Funktionen bekannt sein, kennt aber nicht die aktuellen Bestimmungsparameter: **Alpha**, **Beta** und **Gamma**. Da der Angreifer auch die **Startsequenz** nicht kennt, kann er sie auch nicht herleiten.

**Plaintext --> block key --> XOR-concatenation,**  
**8-bit XOR-concatenation --> 7-bit index values (0...127) und**  
**7-bit index values --> ciphertext array (0...127) --> ciphertext.**



Die Parameter **block key (B Block-Schlüssel)** und **ciphertext array (A Alphabet 128 Bytes)** sind zwei voneinander unabhängige Variable:

$$\text{Cipher text} = f [ f_1 (\text{plain text}, \mathbf{k1}), f_2 (b1, b2), f_3 (b2, \mathbf{k2}) ]$$

$$\text{Plain text} = f [ f_3 (\text{cipher text}, \mathbf{k2}), f_2 (b2, b1), f_1 (b1, \mathbf{k1}) ]$$

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

Beide Prozesse, Verschlüsselungen und Entschlüsselungen, enthalten Gleichungen mit zwei unbekannt Variablen: **k1** and **k2**. Dies führt nur dann zu einem bestimmten Ergebnis, wenn eine Unbekannte aus der anderen abgeleitet werden kann oder wenn zwei Gleichungen mit denselben Unbekannten vorhanden sind. Aber es gibt keine Verbindung zwischen dem in der gleichen Runde generierten Block-Schlüssel = **k1** und Array (Chiffre-Alphabet 128) = **k2**. Beide wurden zwar aus der laufenden CypherMatrix entnommen, haben aber keine funktionale Verbindung untereinander ( $k1 \rightarrow (Hk \text{ MOD } 169)+1$ ) und ( $k2 \rightarrow (Hk+Hp \text{ MOD } 255) +1$ ). Beider gemeinsame Wurzel ist allein die initiale **Startsequenz**. Aber zu dieser Position gibt es keinen Weg zurück (zwei Einweg-Funktionen stehen dagegen).

## B. Hashfunktion

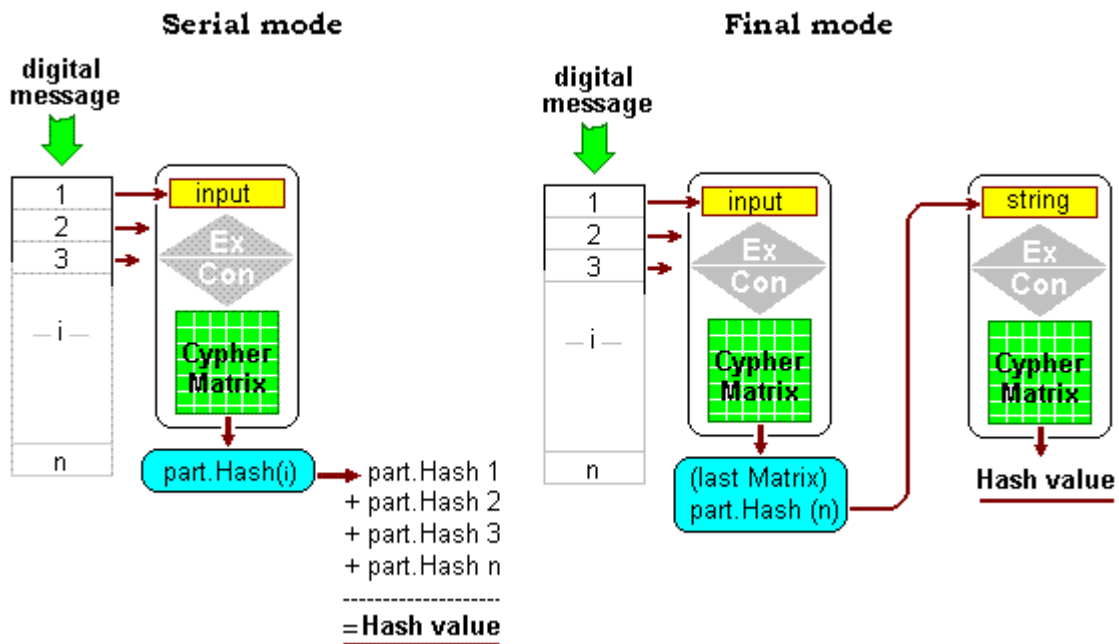
Digitale Sequenzen (Dateien) werden in Klartext-Blöcke (z.B. 64 Bytes) aufgeteilt, die in jedem Durchlauf der **Basis Funktion** nacheinander positionsgewichtet, mit der Hashkonstanten **C(k)** multipliziert, zur Hash-Funktionsfolge erweitert, wieder zur BASIS VARIATION verdichtet und einer dreifachen Permutation unterworfen werden. Die entstehende letzte CypherMatrix mit 256 Elementen stellt den **Hashwert (H)** dar. Ein inhaltsgleicher Hashwert entsteht erst in **256!** (Fakultät) = **8E+506** Fällen. Verglichen mit heute verwendeten Hashfunktionen ergeben sich folgende Vorteile:

1. Eine **Kompression** ist nicht erforderlich,
2. Nur **Bytes** und einfache Mathematik werden verwendet,
3. Mit den Ergebnissen der Funktion (Hashwerte) kann **gerechnet** werden, (z.B. Addition, Subtraktion, Multiplikation, Division und Modulo Rechnung) und
4. die Ergebnisse sind beachtlich **kürzer** als 128 Bit bzw. 160 Bit (SHA, MD5, RIPE-MD).

Grundsätzlich sind zwei Techniken möglich:

- a) Serielle Berechnung partieller Hashwerte in jeder Runde und Addition aller Werte zu einem endgültigen Hashwert (**serial mode**) und
- b) endgültige Berechnung des Hashwerts aus der letzten CypherMatrix nach allen vorhergehenden Durchläufen (**final mode**)

Die Zusammenhänge zeigt das folgende Bild:



Als Beispiel für eine Textdatei >HESSE.TXT< (742 Bytes) errechnet die Funktion im „final mode / last cycle“ folgenden Hashwert:

Serial Matrix Values in HESSE.TXT

---

decimal	base 62	cycle
4541758468324	1HxX1gJw	1
4459228245895	1GVRjHM7	2
4487270920874	1H03XU9S	3
3796561382733	14q7EeZp	4
4295590326511	1DcpPFHj	5
4353281197739	1EdngGll	6
4518558946185	1HYCyqoz	7
3981759374505	186GeYNF	8
4965657775423	1PQEjyUJ	9
5071812250310	1RI6pKi6	10
3970627140060	17u7Gpem	11
194679008757	3QV3HQD	12

---

decimal: 48636785037316  
 base 16: 2C3C224B9004  
 base 62: **DoHE9RTQ**

---

Matrix value last cycle:  
 decimal: 10545443302585169  
 base 16: 2577065A0CB351  
 base 62: **mIU RT1o2b**

---

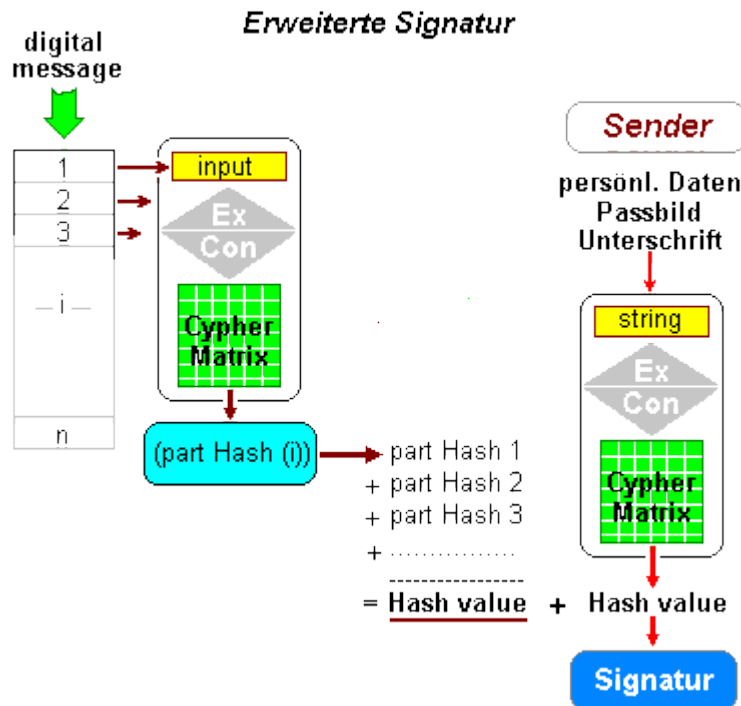
Final **Hash value** for: HESSE.TXT  
 decimal: 10594080087622485  
 base 16: 25A3427C584355  
 base 62: **mWlihBFW1**

=====

Entwicklung und wirkende Faktoren der **CypherMatrix Hashfunktion** werden in einer gesonderten Pdf-Datei dargestellt.

### C. Erweiterte digitale Signatur

Wird die 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.



Als ein individuelles Beispiel folgendes:  
 (Note: Name und Daten sind frei erfunden)

Name, Wohnort: *Karl-Otto Berger, St.Pölten*  
 Geburtstag, Ort: *26.03.1963 / Herzogenaurach*  
 pers. Passphrase: *7 Nordlichter wandern über den großen Belt*  
 Pass Nummer: *1234567-abc-890*

	Basis 62	dezimal
Hashwert der pers. Daten (base 62):	IKEDsWj8t	10333225173361435
pers. Foto (passfoto.jpg): +	12eyYLjGJD	14118070825423503
digitale Unterschrift (mysign.jpg): +	pTlltgq91	11240185700633983
-----		
ID-Daten (Basis 62): =	2dSyY7wpb7	35691481699418921

Wenn die ID-Daten des Anwenders zum Hashwert der Nachricht addiert werden, erhalten wir ein digital signiertes Dokument:

	Basis 62	Basis 16	dezimal
ID-Data Anwender:	2dSyY7wpb7	7ECD36A8338329	35691481699418921
+ Datei Hesse.txt +	mWlihBFW1	+ 25A3427C584355	+10594080087622485
-----			
digitale Signatur	3PzHGp8578	A47079248BC674	46285561787041406

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

- den **Hashwert** der Nachricht errechnen und
- aus der Differenz zur erhaltenen „digitalen Signatur“ die **ID-Daten** des Absenders ermitteln.

Sind die ID-Daten des Absenders in einer Internet-Datenbank (Signatur Portal) gespeichert – selbstverständlich gesichert und verschlüsselt -, kann der Empfänger die Integrität der Nachricht und die Identität des Absenders testen. Die gespeicherten Daten sind so gesichert, dass nur der Inhaber der signierten Nachricht die Informationen abrufen und entschlüsseln kann. Ein Dritter hat keinen Zugriff auf die Daten. Zusätzlich zu den persönlichen Daten des Signierenden werden die digitalen Daten seines Fotos und seiner Unterschrift in den persönlichen Hashwert einbezogen. Der Empfänger der Signatur ist dann in der Lage sowohl die Integrität der Nachricht als auch die Identität des Absenders zu verifizieren. Weitere Einzelheiten in der Pdf-Datei: ["Erweiterte digitale Signatur"](#).

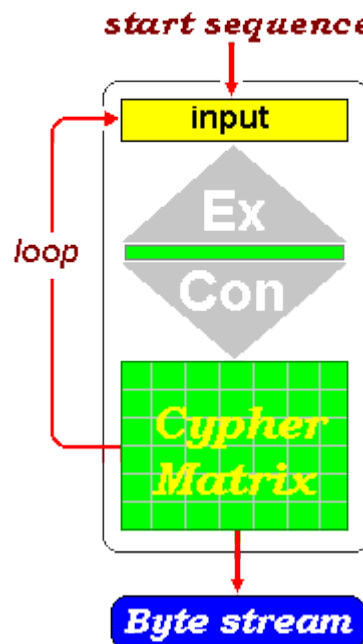
#### D. Unbegrenzte Bytefolgen

Die **Basisfunktion** erzeugt unbegrenzte **Bit-** und **Bytefolgen** (byte stream). In jeder Runde werden 256 Bytes der betreffenden CypherMatrix entsprechend ihrer Verteilung in aufeinander folgenden Zeilen (Bytefolgen) gespeichert. Die Länge dieser Folgen ist vom gewählten Umfang abhängig. Als "**random number series**" sind sie für vielfältige kryptographische Aufgaben einsetzbar (z.B..als Schlüsseldateien für "one-time-pad" und "multi-time-pad" Verschlüsselungen).

Wenn andererseits die **Kontraktion** mit **MODULO 8** (alternativ mit **16, 32, 64** oder andere) vorgenommen wird anstelle MODULO 256 (Kontraktion zur BASIS

VARIATION) können weitere Folgen generiert werden, beispielsweise für Operationen in Verschlüsselungsprogrammen.

### **Byte Generator**



Die bekannten Testprogramme „**ENT.exe**“ und „**FIPS PUB 140-1**“ von U.S.NIST zeigen, dass das Verfahren die meisten Erfordernisse für die Erzeugung von **Zufallsfolgen** erfüllt. Wird auch nur **ein Bit** in der **Startsequenz** geändert, verbleibt in allen weiteren Folgen kein Byte an der gleichen Stelle.

Mehr Einzelheiten über Bytes in kryptographischen Lösungen finden sich im Internet unter:

[www.telecypher.net](http://www.telecypher.net)

Von dort können auch verschiedene DEMO-Programme geladen und selbst getestet werden.

München, im Januar 2010  
**Ernst Erich Schnoor**

---