

# Information

## Neues Verfahren in der Kryptographie

(Ernst Erich Schnoor)

**Informationen** sind eines der wichtigsten Elemente unseres Lebens. Ohne einen sicheren, identischen, authentischen und verbindlichen Informationsaustausch kann eine moderne Wirtschaft nicht optimal arbeiten. Die hierzu in der Informatik entwickelten Verfahren sind zwar geeignet die anstehenden Probleme (Sicherheit, Verbindlichkeit, Integritäts- und Authentizitätsproblem) hinreichend zu lösen, jedoch besteht die Gefahr, wegen der rasanten Fortschritte in der Computertechnik, dass die Sicherheit der gegenwärtigen Verfahren langsam aber sicher schwindet.

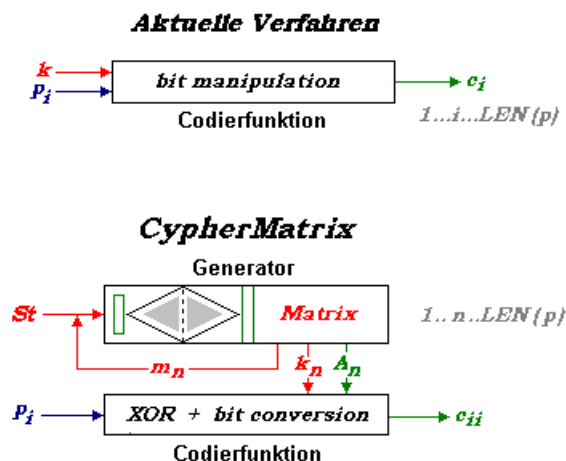
Fast alle heute angewandten Verfahren beruhen auf der **Bit-Technologie**, d.h. sie arbeiten im binären Zahlensystem zur Basis 2 mit den Ziffern „0“ und „1“. Dieser Bereich jedoch dürfte in den vergangenen 30 Jahren bereits völlig erforscht und in allen Teilen zur Genüge bekannt sein. Es ist wie bei einem Ölfeld: wenn alles Öl gefunden ist, gibt das Feld nichts mehr her. Die ständigen „call for papers“ und Ergebnisse der vielen Symposien, Kongresse und Fachveranstaltungen lassen kaum einen anderen Schluss zu. In den meisten Fällen werden hinreichend bekannte Sachverhalte nur noch komplexer. Nach Meinung von Experten (Ron Rivest) erscheint daher ein **Paradigmenwechsel** dringend erforderlich.

Im Folgenden wird ein neues Verfahren vorgestellt, das im Gegensatz zur Bit-Technologie nur mit **Bytes** und mathematischen Verknüpfungen arbeitet (**Byte-Technologie**): das **CypherMatrix Verfahren** (Bezeichnung vom Verfasser).

### CypherMatrix®

Das Verfahren ist in mehrjähriger Forschung entwickelt. Für die Grundlagen hat das DPMA ein Patent erteilt (19811593 vom 28.03.1998). Der Verfasser ist Inhaber aller Rechte.

Abweichend von aktuellen Verfahren umfasst CypherMatrix zwei getrennte Bereiche: **Generator** und **Codierfunktion**. Beide Funktionen können miteinander kombiniert, aber auch einzeln und getrennt voneinander verwendet werden.



Aufgabe des Generators ist die Berechnung von **Hash-Werten (H)** und die Bereitstellung von **Steuerungsparametern** für kryptographische Aufgaben.

Eine ausführliche Darstellung finden Sie im Internet im Web-Artikel:

[Der Kern des CypherMatrix Verfahrens](#)

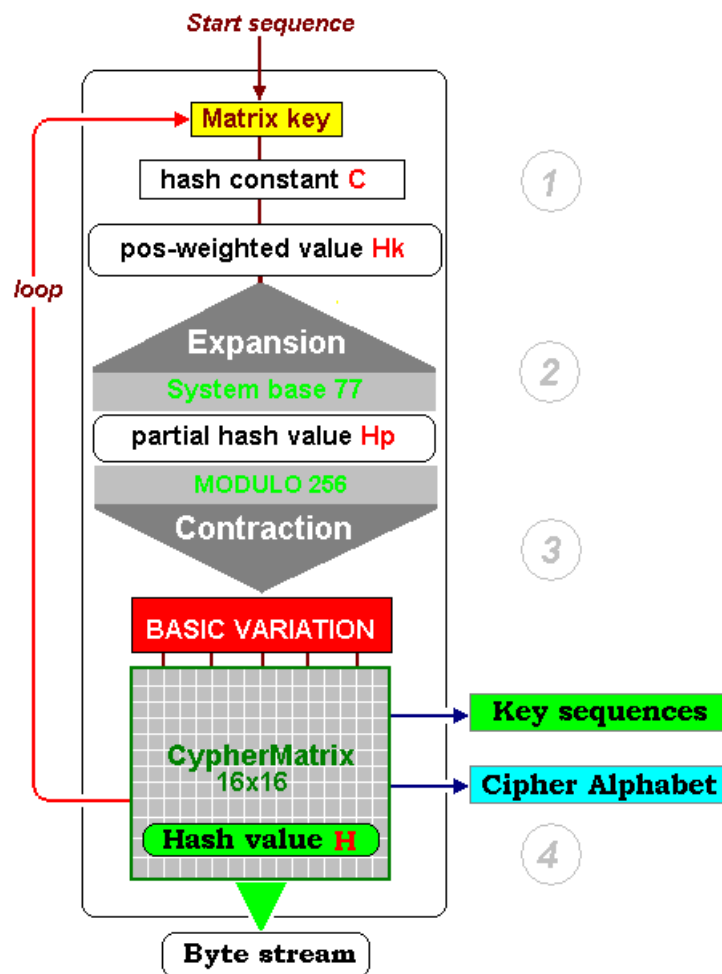
Hier folgt eine kurze Zusammenstellung des Verfahrens:

Eine individuell zu wählende „**start sequence**“ (Passphrase) von 36 bis 64 Bytes (optimal **42** Bytes) initialisiert das Verfahren. Einige Beispiele:

- Im Bodensee werden 147954 Heringe gezählt (41 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)

Eine Passphrase sollte ungewöhnlich aber dennoch leicht zu behalten sein, so dass sie nicht aufgeschrieben werden muss aber auch nicht geraten werden kann. Wegen ihrer Länge und Eigenart ist sie auch keinem Wörterbuchangriff ausgesetzt.

Das folgende Schema zeigt den Aufbau des Generators:



Die **Start-Sequenz** steuert das gesamte Verfahren. Ein Verfahrens-Zyklus arbeitet in vier Stufen:

1. Erweiterung der „start sequence“ zu einem positionsgewichteten Zwischen-Ergebnis (Hk), aufgebaut auf einer Hash-Konstante (C), um Kollisionen zu vermeiden (erste **Ein-Weg-Funktion**),
2. Hochrechnung der Eingabe-Sequenz durch Multiplikation jedes Zeichens zu einer einmaligen Hash-Funktions-Reihe von 160 bis 2400 Zeichen (Expansion) in einem höheren Zahlensystem,
3. Verdichtung der Hash-Funktions-Reihe durch MODULO 256 zu einem Array BASIC-VARIATION von 256 Zeichen (Contraction) (zweite **Ein-Weg-Funktion**),
4. dreifache Permutation der BASIC-VARIATION zur Generierung einer Matrix von 16x16 Elementen als abschließendes Ergebnis.

Die Eingaben – beginnend mit der „**start sequence**“ – bestehen aus Folgen binärer Ziffern „0“ und „1“. Geordnet in 8-Bit Sequenzen als Bytes (**a<sub>i</sub>**) entsprechen sie den dezimalen Werten **0** bis **255** und können als Index-Werte (null bis 255) des erweiterten ASCII-Zeichensatzes verwendet werden. Sie lassen sich außerdem in beliebige Zahlensysteme von zur **Basis 2** bis zur **Basis 128** umrechnen und bearbeiten. Sie sind die Grundlagen der

### Byte-Technologie.

Zur Demonstration der Arbeitsprinzipien des Generators wählen wir die Eingangs-Sequenz mit Worten von Hermann Hesse:

*Als Siddharta den Hain verließ, in welchem der Buddha, der Vollendete, zurückblieb, in welchem Govinda zurückblieb, da fühlte er, dass in diesem Hain auch sein bisherigen Leben hinter ihm zurückblieb und sich von ihm trennte.*

Siddhartha. Eine indische Dichtung, Montagnola 1953

Die „Eingangs-Sequenz“ (hier mit 64 Zeichen) lautet:

**Als Siddharta den Hain verließ, in welchem der Buddha, der Volle**

Die Eingabe-Sequenz (A) ist eine Folge definierter Zeichen (Bytes):

$$A = a_1, a_2, a_3, \dots a_i, \dots a_n$$

Im Folgenden bedeuten:

n = Länge der Eingabe-Sequenz (64 Bytes)

a<sub>i</sub> = Element der Eingabe-Sequenz (erweiterter ASCII-Zeichensatz)

p<sub>i</sub> = Position in der Eingangs-Sequenz

Um der Sequenz (A) univalent einen Wert zuzuordnen, muss jedes Byte (**a<sub>i</sub>**) mit einem Index bewertet und die einzelnen Bytes müssen in geeigneter Weise miteinander verknüpft werden.

Verknüpfung durch Addition:

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

(Die einzelnen Werte für "a" werden um (+1) erhöht, da sonst der Wert ASCII-null keine Berücksichtigung findet)

$$A = \sum_{i=1}^n (a_i + 1)$$
$$A = 5927$$

Der so für (A) errechnete Wert ist allerdings weit davon entfernt als **Hash-Wert** dienen zu können. Zur eindeutigen Verknüpfung der Bytes müssen daher weitere Merkmale hinzukommen. Entsprechend der allgemeinen Aussage, dass jeder Sachverhalt durch Koordinaten für den **Gegenstand**, den **Ort** und die **Zeit** eindeutig bestimmt wird [frei nach: Descartes, René], muss hier zumindest der **Ort** noch zusätzlich berücksichtigt werden. Dabei kann als "Ort" die Position ( $p_i$ ) des Zeichens ( $a_i$ ) in der Sequenz dienen. Die Koordinate Zeit ist nicht relevant ( $t_i=1$ ), es sei denn, zwischen der CPU-Taktfrequenz und bestimmten Bytes besteht eine funktionale Verbindung (eventuell: voice streaming).

Um Kollisionen auszuschließen, berechnet das Verfahren zunächst eine **Hash-Konstante** ( $C_k$ ), die eine Grenze zwischen kollisionsbelasteten und kollisionsfreien Abläufen liefert.

$$n = 64 \text{ (Länge der Eingabe-Sequenz)}$$
$$C_k = n * (n-2) + \text{code}$$
$$\text{code} = 1$$
$$C_k = 3969$$

**Code** ist eine vom Anwender persönlich wählbare Zahl zwischen 1 und 99, mit der die Funktion individualisiert wird. Die Ableitung der Hash-Konstante  $C_k$  wird in der Pdf-Datei [\*\*\*Bestimmungsfaktoren für Kollisionsfreiheit\*\*\*](#) im Einzelnen erläutert.

Eine erste **Ein-Weg-Funktion** errechnet aus der Eingabe-Sequenz (A) einen positionsgewichteten **Hash-Wert** ( $H_k$ ). Dabei wird jedes Zeichen der Start-Sequenz ( $a_i+1$ ) mit seinem Ort ( $p_i$ ) zuzüglich der Hash-Konstante ( $C_k$ ) multipliziert:

$$1 \leq (a_i + 1) \leq 256 \text{ (ASCII-Werte)}$$
$$1 \leq p_i \leq n \text{ (Position von } a_i \text{ in der Eingabe-Sequenz)}$$
$$C_k = \text{Hash-Konstante}$$
$$H_k = \text{positionsgewichteter Hash-Wert}$$

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

$$H_k = 23\,716\,075 \text{ (decimal)}$$

Aber das Zwischen-Ergebnis ( $H_k$ ) ist noch zu klein, um als sicherer Hash-Wert zu bestehen. Zur Erreichung besserer Ergebnisse wird eine zweite **Ein-Weg-Funktion** eingeführt mit einer **Expansion** der Daten in ein höheres Zahlensystem (hier zur **Basis 77**), Errechnung eines weiteren Hash-Wert ( $H_p$ ) und anschließender **Verdichtung** zurück auf dezimale

Werte.

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

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

Die Expansion hat die Aufgabe, die relativ kurzen Eingabewerte (36 bis 64 Bytes) auf möglichst viele Variablen (160 bis 2400 Ziffern, Zeichen) zu erweitern. Jedes Zeichen ( $a_i$ ) wird hochgerechnet auf ( $s_i$ ) und zu einer **Hash-Funktions-Reihe** (Ziffernfolge im Zahlensystem zur Basis 77) verbunden. Die Summe aller erweiterten Werte ( $s_i$ ) bildet dann einen zweiten **Hash-Wert** ( $H_p$ ).

Das Zahlensystem zur **Basis 77** besteht aus folgenden Ziffern:

0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ  
 abcdefghijklmnopqrstuvwxyz&#@àáâãäåæçèéêë

(definiert vom Verfasser, nicht standardisiert)

Auszugsweise entwickeln sich die Daten wie folgt:

Zchn	$a_i + 1$	Pos $i$	$H_k$	Produkt	$+ (p_i + code)$	$s_i$ (Basis 77)
.	....	...	.....	.....	.....	.....
S	84	5	23716075	9960751500	9960751506	3qRJOD
i	106	6	23716075	15083423700	15083423707	5i653i
d	101	7	23716075	16767265025	16767265033	6EêTçv
d	101	8	23716075	19162588600	19162588609	769CHA
h	105	9	23716075	22411690875	22411690885	8Lg4èj
a	98	10	23716075	23241753500	23241753511	8jCJUur
r	115	11	23716075	30000834875	30000834887	B6Xc0N
t	117	12	23716075	33297369300	33297369313	CNGMws
a	98	13	23716075	30214279550	30214279564	BCd297
.	...	..	.....	.....	.....	.....
$H_p =$					4549027780044	L#kTJtt

Erweiterte Daten als **Hash-Funktions-Reihe**

390 Ziffern im Zahlensystem zur **Basis 77**

L#4Furiei@O1â5swT33xêV11C4Cpn3qRJOD5i653i6EêTçv769CHA8Lg4èj8jCJUurB6Xc  
 0NCNGMwsBCd29743r5àFDL7gCáEN2bn@Gf5LâH5Fvld#CBvDgIHDOczSJcxuCaLUcfãâ6o  
 4rkkP1&8ABMQRèK8QFGC#rPyckHæQ0SNgMPâl#WhxVAäxNCHAwOE9JXOS0UnæáJjX5AhvX  
 A9HElabàcMoLX5AhvaaMVdUæYDB3QHazfdAYanUXäæeaãwYhCXPdètççBâHCeGoO5TkQâ4  
 æGDjTqmQSDqZr8oo@jBailëbãcjAAKCâl@kK5âjdB2szLMVCvkFäc0éjng@8g@oçWægvwX  
 ækTeH4gDnGjuqr3GxâHGXRGN3cMyC@UâavF9èjM

Die gewählte Eingangs-Sequenz (64 Bytes) führt zu folgenden Ergebnissen:

Positionsgewichteter Wert ( $H_k$ ): 23 716 075  
 Partieller Hash-Wert ( $H_p$ ): 4 549 027 780 044  
 Gesamter Hash-Wert ( $H_k+H_p$ ): 4 549 051 496 119

Aus den vorstehenden Ergebnissen werden folgende **Steuerungsparameter** abgeleitet::

Variante:  $(Hk \text{ MOD } 11)+1 = 10$  Beginn der Rückumwandlung (78 --> dez)  
 Alpha :  $((Hk+Hp) \text{ MOD } 255)+1 = 215$  Beginn Chiffre-Alphabet (128 Zeichen)  
 Beta :  $(Hk \text{ MOD } 169)+1 = 137$  Beginn Block Schlüssel (63 Zeichen)  
 Gamma :  $((Hp+Code) \text{ MOD } 196)+1 = 78$  Beginn Matrix Schlüssel (42 Zeichen)

Um die Variablen auf dezimale Zahlen zurückzuführen wird eine **Verdichtung** wie folgt vorgenommen: Für die Hash-Funktions-Reihe wird das Zahlensystem **Expansions-Basis +1** unterstellt, hier **Basis 78**. Jeweils 3 Ziffern der Hash-Funktions-Reihe rechnet das Verfahren mit **MODULO 256** zurück in dezimale Zahlen 0 bis 255 (ohne Wiederholung) und speichert sie im Array **BASIC VARIATION** mit 256 Elementen.

L#4FurieI@O1â5swT33xêV11C4Cpn3qRJOD5i653i6EêTçv769CHA8Lg4èj8jCJUrb6Xc  
 0NCNGMwsBCd29743r5àFDL7gCáEN2bn@Gf5LâH5Fvld#CBvDgIHDOczSJcxuCaLUcfââ6o  
 4rkkP1&8ABMQRèK8QFGC#rPyckHæQ0SNgMPâl#WhxVAäxNCHAwoE9JXOS0UnæáJjX5AhvX

Im Vergleich zum Zahlensystem zur Basis 77 enthält die Basis 78 zwar eine zusätzliche Ziffer, aber das beeinträchtigt den Prozess nicht. Die Daten entwickeln sich wie folgt:

Basis 78	dezimal	modulo 256	Array Element
3qR	22335	63	063
qRJ	318493	29 +2	031
RJO	165774	142	142
JOD	117481	233	233
OD5	147035	91	091
D5i	79526	166	166
5i6	33858	66	066
i65	268169	137	137
653	36897	35	035

### Verdichtung der Hash-Funktions-Reihe zum Array **BASIC VARIATION**

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

Da alle Zeichen der **CypherMatrix** Elemente des erweiterten ASCII-Zeichensatzes sind, können die Werte des Arrays BASIC VARIATION direkt in eine Matrix 16x16 übernommen werden. Mit dreifacher Permutation aller Elemente dieser Matrix generiert das Verfahren dann die endgültige CypherMatrix als **finalen Hash-Wert (H)**.

### CypherMatrix

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

Die blau unterlegten Zeichen entsprechen dem **Chiffretext-Alphabet** (ab Alpha = 215). Die CypherMatrix ist eine **eindeutige Abbildung** der Eingabe-Sequenz und stellt somit eine definierte **Hash-Struktur (H)** dar. In jeder Runde wird eine neue CypherMatrix generiert. Nach den Grundsätzen der Wahrscheinlichkeit kommt eine Wiederholung der gleichen Struktur erst in **256!** (Fakultät) = **8E+506** Fällen vor. Das mag unsere Behauptung stärken, die resultierende CypherMatrix sei einmalig und kollisionsfrei.

Zur Steuerung kryptographischer Programme entnimmt das Verfahren der jeweiligen

CypherMatrix definierte Zeichensequenzen an folgenden Positionen:

Variante: (Hk MOD 11)+1	= 10	Beginn der Rückumwandlung (78 --> dez)
Alpha : ((Hk+Hp) MOD 255)+1	= 215	Beginn Chiffre-Alphabet (128 Zeichen)
Beta : (Hk MOD 169)+1	= 137	Beginn Block Schlüssel (63 Zeichen)
Gamma : ((Hp+Code) MOD 196)+1	= 78	Beginn Matrix Schlüssel (42 Zeichen)

### Chiffretext-Alphabet

(Array 128 Zeichen) Alpha = 215

Bei der Entnahme aus der CypherMatrix werden einige Zeichen (Steuerzeichen ASCII-00 bis ASCII-31, ASCII-34, 44 u.a.) ausgeklammert, weil sie in bestimmten Situationen

(z.B. ASCII-26) noch ihren eigentlichen Aufgaben nachgehen und den Ablauf durcheinander bringen.

### Chiffretext-Alphabet

3A F0 AE 74 A3 60 4F 6D 95 46 E9 61 83 67 AD 35  
 53 92 2E 99 6B BD FB 40 41 62 D7 24 CB 70 7B 64  
 CA 8B B8 D6 F1 8E F5 33 57 C7 BA 93 7F B7 F6 3E  
 21 39 4D C4 37 A8 36 AB A9 87 32 86 38 E1 29 5F  
 3D B4 90 9A EA 43 8F D9 7E D4 A1 82 CC 81 EF F2  
 79 45 E4 63 6A 4C D0 3F E5 BC AC 94 E8 96 FA 23  
 5C 7D 52 9C 88 9B 54 85 B3 D5 44 6F 2C A2 E3 49  
 6E BF 9E F9 89 80 A0 73 58 5D 3B EE 2B 78 26 25

(im ASCII-Zeichensatz)

*Index* 1 - 16: : ð ® t £ ` O m • F é a f g 5  
*Index* 17 - 32: S ' . ™ k ½ û @ A b × \$ È p { d  
*Index* 33 - 48: Ê < , Ö ñ Ž ö 3 W Ç ° " □ · ö >  
*Index* 49 - 64: ! 9 M Ä 7 " 6 « © † 2 † 8 á ) \_  
*Index* 65 - 80: = ´ □ š ê C □ Û ~ Ô ¡ , Ì □ ï ò  
*Index* 81 - 96: y E ä c j L Ð ? å ¼ ¬ " è - ú #  
*Index* 97 - 112: \ } R œ ^ > T ... ³ Õ D o , ç ã I  
*Index* 113 - 128: n ¿ ž ù % € s X ] ; î + x & %

Das Chiffretext-Alphabet wird nur in Verschlüsselungsprogrammen benötigt.

### Matrix Schlüssel

(Gamma = 78 --> 42 bytes)

9B 54 85 14 2E B3 D5 20 44 DF B0 B1 13 04 6F 2C DD A2 19 E3 49  
 6E BF 9E F9 89 0E 00 80 A0 73 58 1D 5D 3B EE B2 2B 78 26 25 2D

Der Matrix-Schlüssel dient zur Fortsetzung des Verfahrens. Nach Durchlauf einer Runde wird er auf den Eingang als „**Start Sequenz**“ zur Initialisierung der nächsten Runde zurück geführt (**loop**).

### Block Schlüssel

(Beta = 137 --> 63 bytes)

69 05 84 10 F4 8A BE 47 DB 4E 59 AA 91 18 77 B5 6C 3C 76 D1 EC  
 E2 12 66 98 CF 7A F7 A6 D8 09 DA 4B 8D C9 51 AF E7 ED 55 2A C1  
 A5 71 68 31 D2 E0 E6 50 CE EB 11 34 65 28 8C FC F3 5B CD 15 B6

Der jeweilige Block-Schlüssel dient zur XOR-Verknüpfung mit dem aus dem **Codierbereich** zugeführten gleich langen Klartext-Block. Der Block-Schlüssel wird ebenfalls nur in Verschlüsselungsprogrammen benötigt.

## Anwendungen auf der Grundlage des CypherMatrix Verfahrens

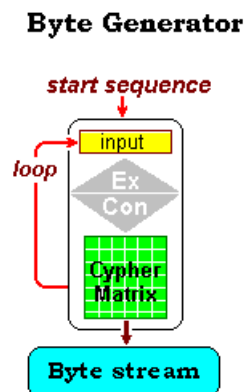
Soweit ersichtlich, ist der Generator neu. Er kann insbesondere eingesetzt werden für:

1. Erzeugung von unbegrenzten Zufallsfolgen (RNG),
2. Dynamische Berechnungen von Hash-Werten,
3. Fortgeschrittene und erweiterte Signaturen,
4. Verschlüsselungen jeden Umfangs und
5. Kontrollfunktionen unterschiedlicher Art.

Für diese Bereiche liegen bereits Demo-Programme in Software vor. Für einen geeigneten kommerziellen Einsatz müssen sie jedoch noch weiter entwickelt werden. Sollte das Verfahren eine entsprechende Akzeptanz erreichen, wird sich eine Implementierung in Hardware als **CypherChip** anbieten (vielleicht als Gegenpol zur heute dominierenden Crypto-Industrie, die noch weitgehend auf herkömmlicher Bit-Technologie aufbaut).

### A. Random Byte Generator

Infolge Übernahme der Zeichen aus der jeweils vorhergehenden CypherMatrix wirken sich die Permutationen der Elemente auf alle folgenden Runden aus. Da für alle Zeichen die gleiche Wahrscheinlichkeit besteht, dass sie „weitergereicht“ werden, kommen hier starke Zufallsmomente für die Verteilung der Zeichen in der jeweils nächsten CypherMatrix ins Spiel. Da der gesamte Verlauf allein durch die **Start-Sequenz** der ersten Runde gesteuert wird, kann diese Eigenschaft für die Erzeugung einer **unbegrenzten Bytefolge** als „random number generator“ genutzt werden.

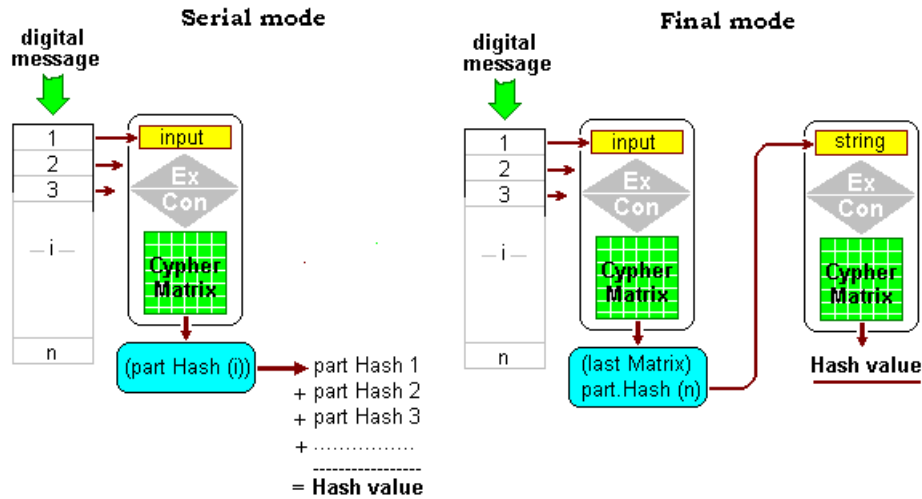


Einzelheiten hierzu finden Sie im WEB-Artikel: *Cypher's Kern* ["Random Byte Generator"](#)

### B. Dynamische Hash-Funktion

Digitale Zeichenfolgen beliebiger Länge (Dateien) werden in Klartext-Blöcke von z.B. 64 Bytes (wahlweise von 36 bis 64 Bytes) geteilt, die das Verfahren dann in seriellen Runden abarbeitet und für jede Runde eine eigene CypherMatrix (16x16) als **Hash-Wert** errechnet.

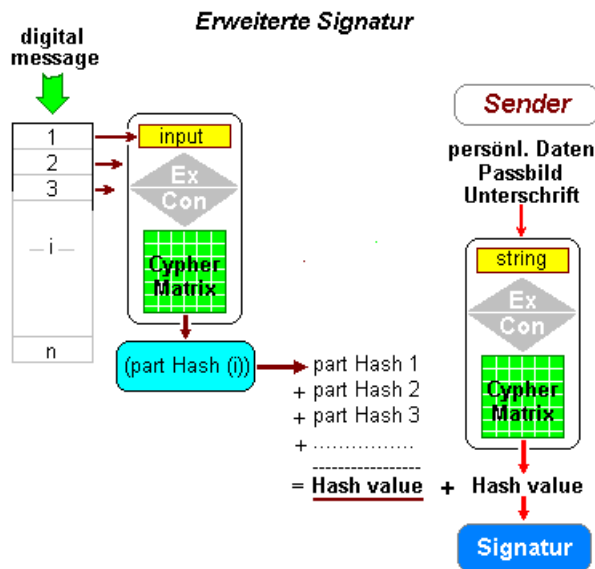
Dabei sind grundsätzlich zwei Methoden möglich:



Im Gegensatz zu den heute angewendeten Hashfunktionen ist eine Kompressionsfunktion nicht erforderlich. Entwicklung und Arbeitsweise der „CypherMatrix Hash-Funktion“ sind in der pdf-Datei [Dynamische Hash-Funktion](#) dargelegt.

### C. Erweiterte Signatur-Funktion

Aus der kombinierten Anwendung des Generators als Hash-Funktion, sowohl auf eine zu signierende **digitale Nachricht** als auch auf die **persönlichen Daten** des Signierenden (Sender) einschließlich Bild und Unterschrift kann eine „**einfache Signatur**“ erzeugt werden.



Einzelheiten und praktische Durchführung werden in der Datei [Signatur.pdf](#) geschildert.

## D. Durchführung von Verschlüsselungen

Das **CypherMatrix** Verfahren bewirkt die Verschlüsselung mit der Kombination dreier unterschiedlicher Operationen: **XOR-Verknüpfung**, **Substitution „dyn24“** und **Bit-Konversion**:

### Substitute



Modifikationen der  
Bit Konversion:

1. 8 bit → 6 bit
2. 8 bit → 7 bit
3. 8 bit → 8 bit
4. 8 bit → 9 bit

Die XOR-Verknüpfung ist eine Standardoperation. Da aber bei einfachem XOR noch kein wirklicher Schutz gegeben ist, verwendet das Verfahren zwei weitere Verschlüsselungsschritte: die Substitution „**dyn24**“ und als entscheidende Neuerung die vom Autor so genannte „**Bit-Konversion: C1,2,3,4**“.

Mit den drei Operationen lassen sich folgende Kombinationen gestalten:

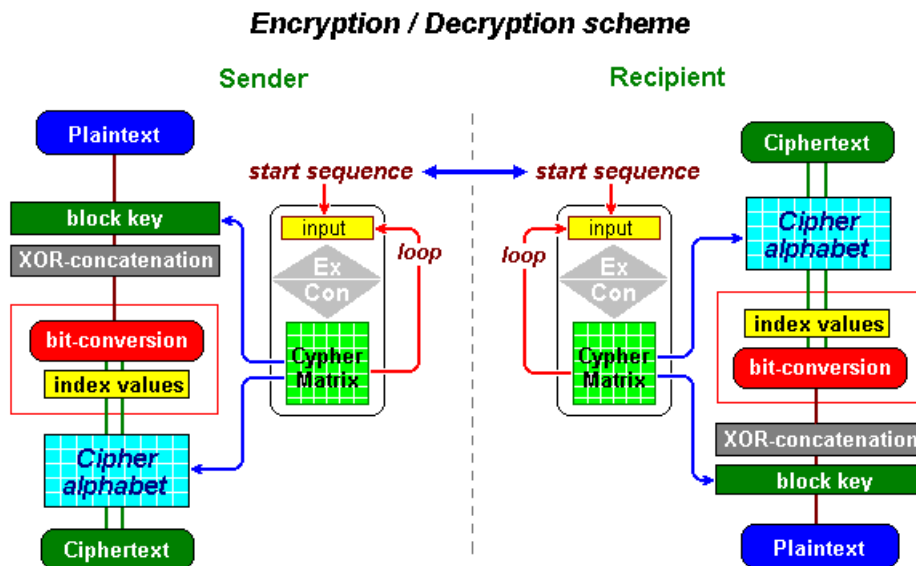
<i>Einfache Kombination</i>	
$P \text{ --- } X$	CYPHER11
$P \text{ --- } Y$	CYPHER12
$P \text{ --- } C1,2,3,4$	CYPHER13
<i>Zweifache Kombination</i>	
$P \text{ --- } X \text{ --- } Y$	CYPHER21
$P \text{ --- } X \text{ --- } C1,2,3,4$	CYPHER22
$P \text{ --- } Y \text{ --- } X$	CYPHER23
$P \text{ --- } Y \text{ --- } C1,2,3,4$	CYPHER24
$P \text{ --- } C1,2,3,4 \text{ --- } X$	CYPHER25
$P \text{ --- } C1,2,3,4 \text{ --- } Y$	CYPHER26
<i>Dreifache Kombination</i>	
$P \text{ --- } X \text{ --- } Y \text{ --- } C1,2,3,4$	CYPHER31
$P \text{ --- } X \text{ --- } C1,2,3,4 \text{ --- } Y$	CYPHER32
$P \text{ --- } Y \text{ --- } X \text{ --- } C1,2,3,4$	CYPHER33
$P \text{ --- } Y \text{ --- } C1,2,3,4 \text{ --- } X$	CYPHER34
$P \text{ --- } C1,2,3,4 \text{ --- } X \text{ --- } Y$	CYPHER35
$P \text{ --- } C1,2,3,4 \text{ --- } Y \text{ --- } X$	CYPHER36

48 Kombinationen

Als Klartext **P** wird grundsätzlich jedes Byte erfasst, das sind insbesondere alle 256 ASCII-Zeichen. Die Verschlüsselung – beispielsweise **CYPHER22** – vollzieht sich in drei Funktionen:

1. Partielles dynamisches „One-time-pad“:  
*Klartext --- Schlüssel --- XOR-Verknüpfung,*
2. Bit-Konversion:  
*8-bit XOR Verknüpfung --- 7-bit Indexwerte (0...127) und*
3. Bestimmung des Chiffretextes:  
*7-bit Indexwerte --- Chiffre-Array --- Chiffretext.*

**Klartextblöcke** (z.B. 63 Byte) und die aus der CypherMatrix entnommenen zugehörigen **Blockschlüssel** (ebenfalls 63 Byte) haben stets die gleiche Länge. Beide werden XOR-verknüpft. Das entspricht im Prinzip einem dynamischen „one-time-pad“, jedenfalls für jede einzelne Runde. Das folgende Schema vermittelt einen Eindruck der Zusammenhänge:



**>Substitution „dyn24“<**

Bei der alternativen Substitution „dyn24“ (Bezeichnung von Autor) werden aus der **BASIC VARIATION** durch zyklische Permutationen zwei **vierdimensionale** Matrixen (2x2x2x2 Zeichen): **Matrix A** und **Matrix B** gebildet. In jeder Matrix sind alle einstelligen hexadezimalen Ziffern (**0 – F**) gut durchmischt gespeichert. Vor jedem Zyklus von z.B. **63** Klartextzeichen werden die Ziffern permutiert und Matrix A und Matrix B neu gebildet.

**BASIC-VARIATION (16 Elemente)**

Matrix A: 0 2 12 15 4 11 13 14 1 3 5 6 7 8 9 10

Matrix B: 10 0 12 1 5 4 15 6 14 7 3 8 9 11 2 13

**(2 Matrixen mit je 4 Dimensionen)**

Matrix A	Matrix B
0 C 4 D 1 5 7 9	A C 5 F E 3 9 2
2 F B E 3 6 8 A	0 1 4 6 7 8 B D

Zur **Substitution** eines Klartextzeichens nimmt das Programm dessen 8-bit Sequenz in hexadezimaler Notation (zweistellig: 00 bis FF), sucht die **erste** Stelle in der **Matrix A**, die **zweite** Stelle in der **Matrix B** und verbindet die je vier gefundenen **Indizes** (-1) (0000 bis 1111) zu einer **achtstelligen Binärzahl**. Diese Binärzahlen stellen dann die Substitution des jeweiligen Klartextzeichens dar.

### >Bit-Konversion<

Die **Bit-Konversion** stellt datentechnisch einen Wechsel in der Anzahl der Bits in dem entsprechenden Zeichen dar. Die Folge der 8-Bit Sequenzen beispielsweise aus der XOR-Verknüpfung (0-255) werden in 7-Bit Abschnitte (0-127) unterteilt und als Indizes (+1) für die Zeichen im **Chiffre-Alphabet** verwendet. Kein Bit wird hinzugefügt und kein Bit entfernt. Die Reihenfolge der Ziffern „0“ und „1“ bleibt unverändert, es wird nur eine Umgruppierung (8-Bit --> 7-Bit) vorgenommen.

8-Bit XOR-Sequenzen umgruppiert in 7-Bit Sequenzen als "Index-Werte"

8-bit: 11101111100001111001101010101111101010011101111101000100 ....  
 7-bit: 11101111100001111001101010101111101010011101111101000100 ....

Eine ausführliche Darstellung der Bit-Konversion wird in der Pdf-Datei [XOR und Bit-Conversion](#) gegeben.

Infolge **Bit-Konversion** verlängert sich der Chiffretext gegenüber dem Klartext im **Verhältnis 8:7**. Damit wird auch die funktionale Verbindung vom Klartext zum Chiffretext durchbrochen. Es kann nicht mehr jeder Klartextbuchstabe mit einem zugehörigen Chiffretext-Zeichen in Beziehung gesetzt werden. Das hat entscheidende Auswirkungen auf die Kryptanalyse des Verfahrens.

## E. Kryptanalyse des CypherMatrix Verfahrens

Die heute praktizierten Angriffe auf verschlüsselte Nachrichten setzen fast alle voraus, dass Klartext und Chiffretext die gleiche Länge haben (**Längenkongruenz**). Klartext und Chiffretext haben grundsätzlich die gleiche Anzahl von Zeichen (Padding ausgenommen). Für jedes Klartextzeichen ist ein bestimmtes Chiffretextzeichen vorhanden, sonst könnte beispielsweise ein Strukturvergleich auch gar nicht vorgenommen werden.

Da beim CypherMatrix Verfahren der Chiffretext stets um **8/7** länger ist als der Klartext, können die üblichen Angriffe – wie Strukturanalysen, known plaintext attack, chosen plaintext attack, differenzielle und lineare Kryptoanalyse u.a. – wegen der fehlenden Längenkongruenz nichts ausrichten und wir können sie vergessen. Die „Bit-Konversion“ durchbricht die funktionale Beziehung zwischen Chiffretext und Klartext. Und sogar „**brute force**“ ist in Frage zu ziehen. Lesen Sie hierzu den diesbezüglichen Abschnitt in „**Cypher's Kern**“ [Brute force attack](#)

Bei einem Angriff auf die Start Sequenz (36 bis 64 Bytes) ergibt eine Länge von **42 Bytes** bei einer **Entropie** von **336** eine exponentielle Komplexität von  $O(2^{336}) = 1.4E+101$ . Im Fall einer 64 Bytes langen Start Sequenz beträgt die Schlüssellänge sogar 512 Bytes und

die Komplexität folglich  $O(2^{512}) = 1.34E+154$ . Ein „brute force“ Angriff ist hier aussichtslos.

Für einen „brute force“ Angriff im laufenden Verfahren besteht grundsätzlich kein Einstiegspunkt. Ein „brute force“ Angriff vom Ende her kann aus folgenden Gründen keinen Erfolg haben: Dem Angreifer sind grundsätzlich nur der **Chiffretext** und dessen **Länge** bekannt. Außerdem kennt er das CypherMatrix Verfahren mit seinen drei Funktionen:

```
(f1): Klartext --> Schlüssel --> 8-bit XOR-Sequenzen  
(f2): 8-bit XOR-Sequenzen --> 7-bit Index-Werte  
(f3): 7-bit Index-Werte --> Array (128) --> Chiffretext
```

Die Parameter **Schlüssel** und **Array (128)** sind zwei voneinander unabhängige Variable.

$$c_m = f [ f_1 (p_n, k_1), f_2 (b_1, b_2), f_3 (b_2, k_2) ]$$
$$p_n = f [ f_3 (c_m, k_2), f_2 (b_2, b_1), f_1 (b_1, k_1) ]$$

$f_x$  = funktionale Verbindung  
 $p_n$  = Klartext  
 $k_1$  = **Schlüssel**  
 $b_1$  = 8-bit Sequenz  
 $b_2$  = 7-bit Index-Wert  
 $k_2$  = **Array (128)**  
 $c_m$  = Geheimtext

Die Ermittlung des Chiffretextes  $c_m$  und die retrograde Suche nach dem Klartext  $p_n$  zeigen sich somit als Gleichungen mit zwei **unbekannten** Veränderlichen:  $k_1$  und  $k_2$ . 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. Es gibt viele Paare **Array / Schlüssel**, die nach einem versuchten "brute force" Angriff irgendwelche lesbaren **Klartexte** liefern, von denen man jedoch nicht weiß, welcher der Richtige ist.

Zwischen dem jeweiligen **Schlüssel** und dem in derselben Runde generierten **Array (128)** (Geheimzeichen-Alphabet) gibt es keine Verbindung. Ihre gemeinsame Wurzel liegt allein in der ursprünglichen **Start-Sequenz**. Aber dahin führt kein Weg zurück (Einwegfunktionen). Folglich muss auch „brute force“ scheitern.

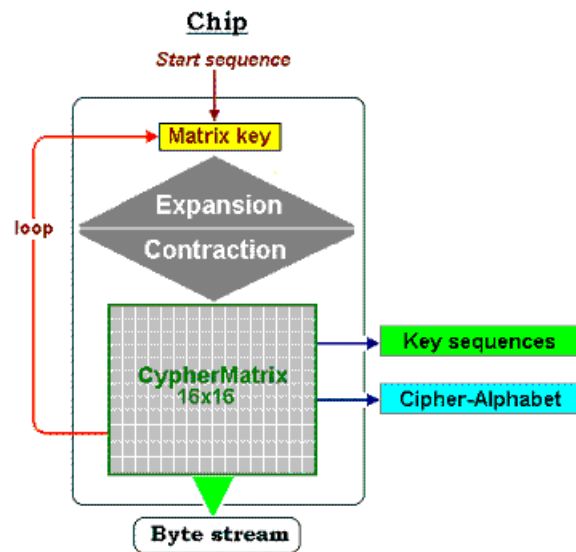
## F. Schlussfolgerungen

Wenn alle herkömmlichen Angriffe gegen das **CypherMatrix** Verfahren infolge **Bit-Konversion** und fehlender "**Längenkongruenz**" nicht zum Erfolg führen und auch der verbleibende "**brute force**" Angriff aussichtslos ist, dann ergibt sich nur eine Schlussfolgerung:

*Das Verfahren ist **sicher** und **nicht brechbar**.*

Widersprüche gegen diese Feststellung werden jederzeit entgegen genommen und ausführlich geprüft.

Als nächste Aufgabe bliebe die Entwicklung eines Plattform unabhängigen **Chips** mit den Funktionen des **CypherMatrix** Verfahrens, der universell in kryptographischen Abläufen eingesetzt werden könnte.



Aber diese Aufgabe übersteigt die Möglichkeiten des Verfassers. Bisher hat sich allerdings noch kein kompetenter Partner gefunden.

### Link-Übersicht

[Start-Seite "CypherMatrix"](#)

[Byte-Technologie](#)

[Cypher's Kern](#)

[Längenkongruenz](#)

[CypherMatrix Hash-Funktion](#)

[Erweitertes Signatur-Verfahren](#)

[Kontroll-Verfahren](#)

[XOR und Bit-Konversion](#)

[Kollisionsfreier Verlauf](#)

[Download](#)

München, im Mai 2007

**Der Verfasser**

© Copyright

Diplomkaufmann

Ernst Erich Schnoor

---