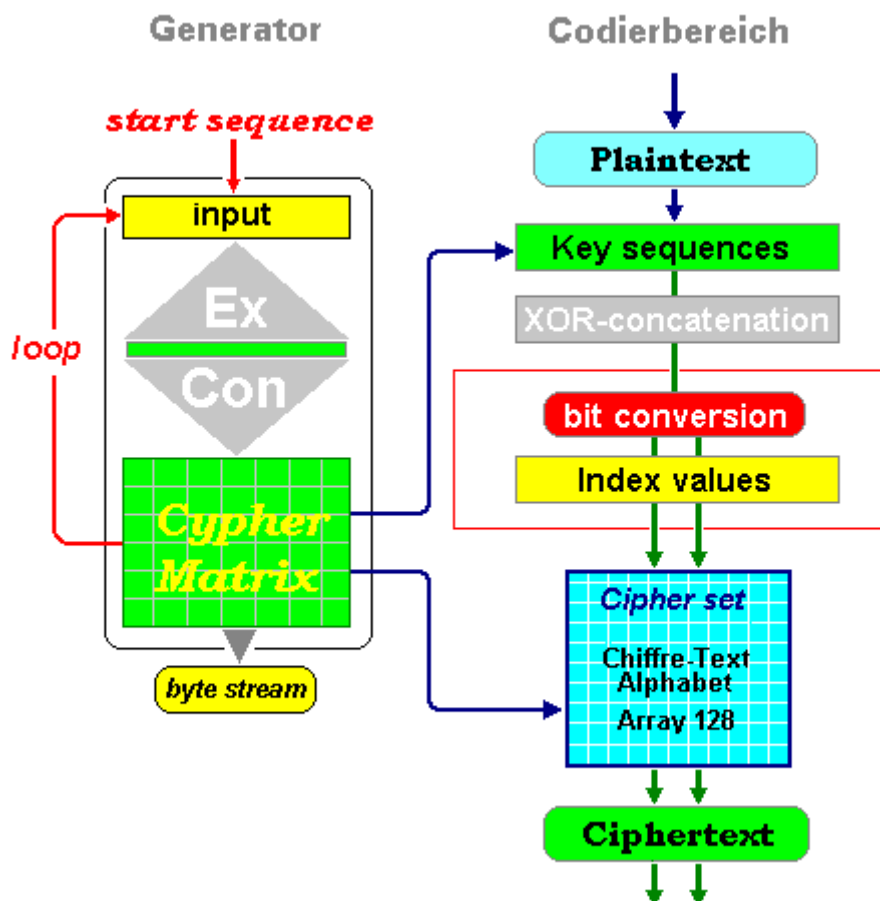


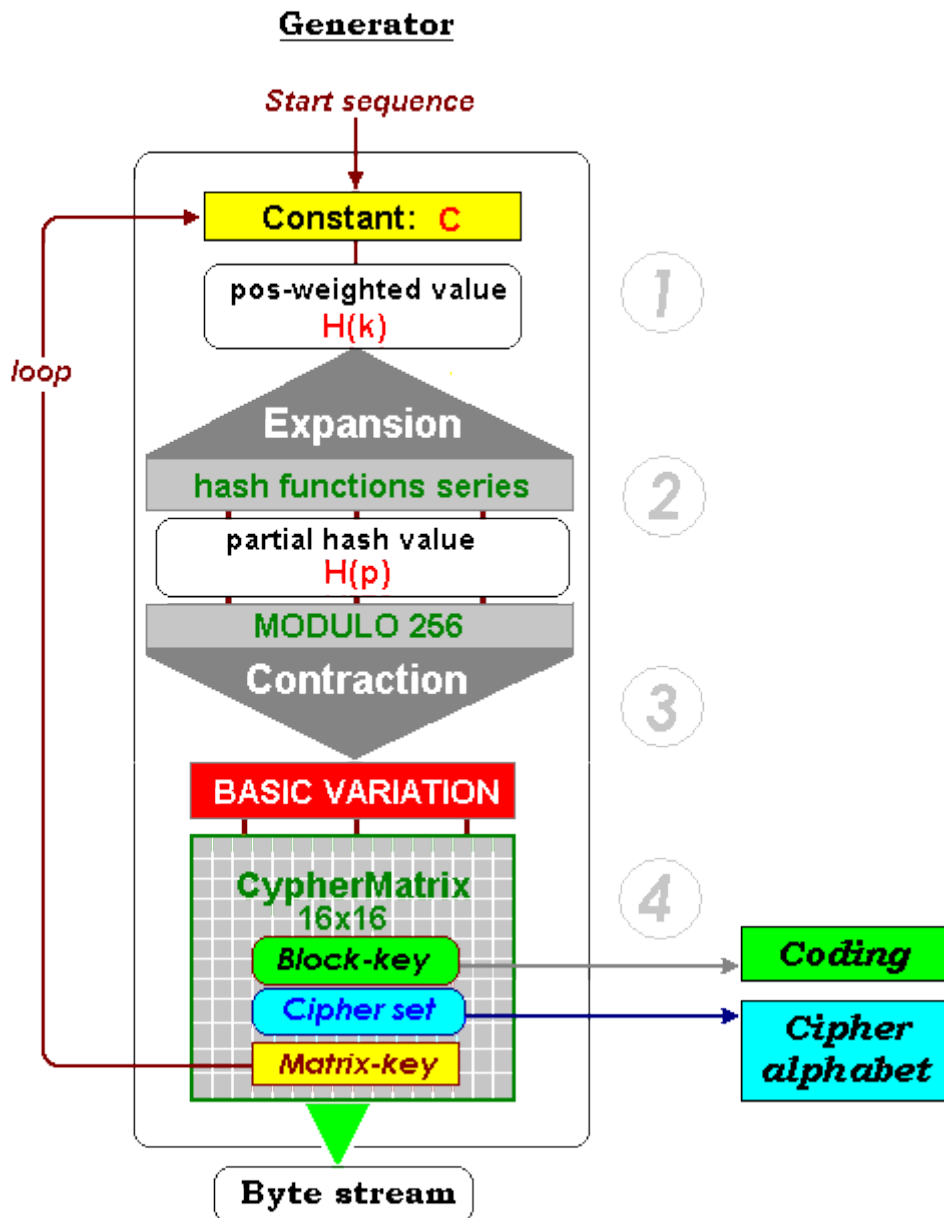
CypherMatrix

Aufbau des Daten Generators

CypherMatrix arbeitet mit zwei getrennten Bereichen:



Beide Bereiche werden miteinander kombiniert, können aber auch getrennt voneinander eingesetzt werden.



Die CypherMatrix liefert alle Steuerungsparameter, die für den Ablauf kryptographischer Anwendungen erforderlich sind, insbesondere den **Matrix-Schlüssel**, eine Folge von 42 Bytes, die als neue Start-Sequenz für den nächsten Durchgang auf den Eingang zurück gekoppelt wird (loop).

Der Generator arbeitet in 4 Stufen:

1. Positionsgewichtung $H(k)$ der Start-Sequenz und Einbau der Hash-Konstante $C(k)$ zum kollisionsfreien Verlauf,
2. **Expansion**: Hochrechnung zur Hash-Funktions-Reihe im Zahlensystem zur **Basis 77** und Zwischenwert $H(p)$,
3. **Contraction**: Verdichtung der Hash-Funktions-Reihe mit Modulo 256 zur **BASIC-VARIATION** und
4. Dreifache Permutation zur CypherMatrix (finaler **Hash-Wert H**).



Erweiterung der **Start-Sequenz** zum positionsgewichteten **Zwischen-Wert H_k**

Alle digitalen Daten sind Folgen der Ziffern „0“ und „1“ (bits). Aufgeteilt in 8-bit Sequenzen (bytes) lassen sich die Daten in dezimalen Zahlen (0 bis 255) ausdrücken, mit denen gerechnet werden kann, die in andere Zahlensysteme (Basis 3 bis Basis 256) umgerechnet und die als Indexwerte dem erweiterten ASCII-Zeichensatz (null bis 255) zugeordnet werden können.

Die Start-Sequenz ist eine Folge von definierten Bytes (a_i) der Länge (n).

$$\text{Sequenz} = a_1 \ a_2 \ a_3 \ \dots \ a_i \ \dots \ a_n$$

Für die folgenden Erläuterungen wählen wir die Start-Sequenz:

Bruno der Braunbär aus Bregenz im Breisgau

Um der Sequenz ($n = 42$) univalent einen Wert $H(k)$ zuzuordnen, muss jedes Byte (a_i) mit einem Wert definiert und die einzelnen Bytes müssen in geeigneter Weise miteinander verknüpft werden. Als Werte wählen wir die Indexwerte des erweiterten ASCII-Zeichensatzes.

Verknüpfung durch Addition:

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

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

$$H(k) = \sum_{i=1}^n (a_i + 1)$$
$$H(k) = 3993$$

Der Wert für $H(k)$ ist zu klein, um eine eindeutige Abbildung der Start-Sequenz darzustellen. Es müssen weitere Differenzierungen vorgenommen werden.

In Anlehnung an **Renè Descartes** kann jeder Sachverhalt – vorausgesetzt er ist in seinen Dimensionen skalierbar - durch seine Koordinaten für den **Gegenstand**, den **Ort** und die **Zeit** eindeutig bestimmt werden (kartesisches Koordinatensystem).

„Gegenstand“ ist das einzelne Zeichen **a(i)**. Die „Zeit“ ist nur dann relevant, wenn zwischen CPU-Taktfrequenz und einzelnen Bytes auch eine funktionale Verbindung besteht. Die ist hier jedoch konstant. Die Koordinate „Zeit“ wird daher mit **t = 1** angesetzt. Als „Ort“ bietet sich die Position **p(i)** des Zeichens **a(i)** in der Eingabe-Sequenz an.

Jedes Zeichen **a(i)** wird in der Weise **positionsgewichtet**, indem es mit seinem „Ort“ **p(i)** verknüpft, d.h. multipliziert wird. Andere Verknüpfungen sind durchaus denkbar.

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

$$t_i = 1$$

Das Auftreten von Kollisionen ist allerdings noch nicht ausgeschlossen. Es muss daher zusätzlich ein Faktor eingebunden werden, der Kollisionen grundsätzlich ausschließt. Diese Aufgabe übernimmt die **Hashkonstante C(k)** (Bezeichnung vom Autor). Sie ist nur von der Länge (**n**) der Eingabe-Sequenz und einem individuell einzugebenden „Anwender-Code“ (1 bis 99) abhängig.

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

$$C(k) = 1680 + 1 = 1681$$

Im vorliegenden Beispiel ist der **code** mit **1** festgelegt.

Die Ableitung der Hashkonstante **C(k)** ist dargelegt im Internet unter:

<http://www.telecypher.net/Kollfrei.pdf>

Unter Einbeziehung der Hashkonstanten **C(k)** errechnet sich das partielle Ergebnis **H(k)** dann wie folgt:

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

$$H_k = 6798793$$

Der errechnete Wert für **H(k)** schließt zwar Kollisionen aus, ist aber für einen unangreifbaren Hash-Wert immer noch zu niedrig. Um sichere Ergebnisse zu erzielen, wird zusätzlich eine **Expansion** durchgeführt, die ohne zusätzliche Eingaben die Bestimmungsfaktoren auf eine möglichst große Zahl auffächert ohne die Eigenschaft der Kollisionsfreiheit zu verlieren.

2

Hochrechnung der **Start-Sequenz** zur **Hash-Funktions-Reihe** im **Zahlensystem** zur **Basis 77 (Expansion)** und **Hash-Wert H_p**

Die Expansion hat die Aufgabe, die Eingabewerte von 42 Bytes in ihrer bisherigen Form auf möglichst viele Variable (etwa 160 bis 2400) hoch zu rechnen. Dazu werden die dezimalen Werte in ein höheres Zahlensystem umgewandelt – hier zur **Basis 77** - und mit dem jeweiligen Ergebnis S_i in einer Hash-Funktions-Reihe gespeichert. Gleichzeitig errechnet das Verfahren die Summe aller umgerechneten Werte S_i als weiteren partiellen Zwischenwert H_p . Anstelle der Basis 77 können auch andere Zahlensysteme verwendet werden, beispielsweise von zur Basis 36 bis zur Basis 128.

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

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

Das gewählte Zahlensystem zur **Basis 77** hat folgende Ziffern:

0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz&#@àáâãäåæçèéë

(definiert vom Autor, nicht standardisiert)

EXPANSION

Die **Hash-Funktions-Reihe** umfasst 501 Ziffern im Zahlensystem zur **Basis 77**

CèxëäibGQ2ãZàOo18âBY@1VNn#RcMoãp1xuzçM23#4Ut2kDZ1i##XbX1àfhêu3Zäl273FU
rpu4Bcè#o4E1gç@3êRM9z5qMP1k5FQâ3A1iKW#V4æ5wâu6HJçla6VhéBY1ázFWD42ërëG7
H3Btq6oâEMz746St@7DSFUa86i62F9KpOWP2hàvks8e2XTb994yPj2#06425ãfKçæAUr#6
J9aâKQdA92ãd#BRéXYLAYhp8hA77écuCYdâaæ2#WT3âYæaâdYCucé77Ah8phYALYXéRB#d
ã29AdQKåa9J6#rUAæçKfã52460#2jPy499bTX2e8skvâh2PWOpK9F26i68aUFSD7@tS647
zMEào6qtB3H7Gérë24DWFzä1YBéhV6alçJH6uâw5æ4V#WKi1A3âQF5k1PMq5z9MRê3@çg1
E4o#ècB4uprUF372LäZ3uêhfà1XbX##i1ZDK2tU4#32Mçzux1pãoMcR#nNV1@YBâ81oOàZ
ã2QGbiäèxèC

Die Expansion stellt die erste **Einweg-Funktion** des Verfahrens dar. Es gibt keinen Weg zurück zur Start-Sequenz. Die Funktion bewirkt außerdem, dass die Start-Sequenz nur als Ganzes gesucht und gefunden werden kann, wenn überhaupt.

Die Durchführung der Expansion führt zu folgenden Bestimmungsfaktoren:

	dezimal	Basis 77
Hashkonstante C(k):	1681	L@
positionsgewichteter Wert (H _k):	6798793	EπS1
partieller Zwischenwert (H _p):	588503523025	2#WECDX
gesamter Hashwert (H _p +H _k):	588510321818	SIFxbqz

Daraus errechnet das Verfahren die folgenden Steuerungsparameter:

Variante	(H_k MOD 11) +1	= 2	Beginn Rückumwandlung
Alpha	((H_k + H_p) MOD 255) +1	= 249	Offset Chiffre-Alphabet
Beta	(H_k MOD 169) +1	= 93	Offset Block-Schlüssel
Gamma	((H_p + code) MOD 196) +1	= 7	Offset Matrix-Schlüssel
Delta	((H_k + H_p) MOD 155) +1	= 144	dynamische Bit-Folgen
Theta	(H_k MOD 32) +1	= 10	dynamische Zahlenfolgen

In die Berechnung von **Gamma** (Abgriff für den Matrix-Schlüssel) und **Delta** fließt der vom Anwender gewählte **Code** ein. Bei gleicher Start-Sequenz können damit 99 verschiedene Abläufe initiiert werden. Da der **Matrix-Schlüssel** als Eingangs-Sequenz für den nächsten Durchlauf verwendet wird, ergeben sich in jeder Runde für die Steuerungsparameter völlig unterschiedliche Werte.

Die **Hash-Funktions-Reihe** errechnet sich auszugsweise wie folgt:

char	a _{i+1}	p _i	(a _{i+1})*p _i	H _k	S _i	Basis 77
B	67	11	737	6798793	5010710453	1àfhêu
r	115	12	1380	6798793	9382334353	3Zäl27
a	98	13	1274	6798793	8661662296	3FU rpu
u	118	14	1652	6798793	11231606051	4Bcè#o
n	111	15	1665	6798793	11319990361	4E1gç@
b	99	16	1584	6798793	10769288129	3êRM9z
ä	133	17	2261	6798793	15372070991	5qMP1k
r	115	18	2070	6798793	14073501529	5FQâ3A
...
			Summe H _p	588503523025	2#WECDX	

Um die Variablen der Hash-Funktions-Reihe wieder auf dezimale Größen zurückzuführen, vollzieht das Verfahren anschließend als dritte Stufe eine **Kontraktion** der Bestimmungsgrößen auf Zahlen im dezimalen Zahlensystem.

3

Verdichtung der Hash-Funktions-Reihe durch Modulo 256 zum Array **BASIC-VARIATION** (Contraction)

Aus der **Hash-Funktions-Reihe** werden in jeder Runde jeweils drei Ziffern der Reihe mit **MODULO 256** in dezimale Zahlen **0** bis **255** (ohne Wiederholung) umgewandelt. Vom Ergebnis wird die Variable **Theta** abgezogen und der verbleibende Wert als **BASIC-VARIATION** in einem Array mit 256 Elementen gespeichert. Bei der Rückumwandlung wird für die Ziffern der Hash-Funktions-Reihe das Zahlensystem zur **Basis 78** unterstellt (Expansionsbasis **77 + 1**). Im Vergleich zur Basis 77 enthält die Basis 78 zwar eine zusätzliche Ziffer, aber das beeinträchtigt den Prozess nicht.

Der Prozess stellt die zweite **Einweg-Funktion** dar. Die Funktion ist nicht umkehrbar und eine Bestimmung der Hash-Funktions-Reihe retrograd ist nicht möglich. Der Kontraktionsprozess kann alternativ auch mit anderen MODULO-Faktoren (insbesondere 8, 16, 24, 32 oder 64) durchgeführt werden. So lassen sich beispielsweise mit MODULO 64 herkömmliche **S-Boxen** generieren.

In Pseudo-Code Darstellung geschieht folgendes:

```
p = Variante
FOR k=1 TO 256
  Zahl = MID$(Reihe, p, 3)
  CALL SystemNachDez (78, Zahl, Dezimal)
  Element = (Dezimal MOD 256)
  INCR p
  Variation(k)=Element
  IF k>1 THEN
    n = 0
    DO
      INCR n
      IF Variation(n) = Element THEN
        INCR Element
        Variation(k) = Element
        n = 0
      END IF
    LOOP UNTIL n = k-1
  END IF
NEXT k
Index = Variation(k) - Theta
IF Index<0 THEN Index = 256 + Index
Variation(k) = Index
```

dreistellige Zahl Basis 78
Rückumwandlung
Begrenzung auf 0 bis 255

BASIC-VARIATION: 256 Elemente

Mit dem Parameter **Variante = 2** (Beginn der Rückumwandlung) beginnt die Rückrechnung an der zweiten Position der Hash-Funktions-Reihe.

Danach ergibt sich für unser Beispiel folgende Entwicklung der Daten:

CèxëäibGQ2äZàOo18âBY@1VNn#RcMoãp1xuzçM23#4Ut2kDZ1i##XbX

3 Ziffern Basis 78	dezimal	modulo 256	- Theta	Element
èxë	448810	42	10	32
xëä	364953	153	10	143
ëäi	467810	98	10	88
äib	423265	97	10	87
ibG	270598	6	10 (256-4)	252
bGQ	226382	78	10	68
GQ2	99374	46	10	36
Q2ä	158408	200	10	190

BASIC-VARIATION (256 Elemente)
Verteilung der Elemente

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



**dreifache Permutation der BASIC-VARIATION
zur CypherMatrix als finaler Hash-Wert H**

Die **BASIC-VARIATION** ist eine eindeutige nicht umkehrbare Abbildung der Start-Sequenz bzw. des fortgeschriebenen Matrix-Schlüssels. Die Werte der BASIC-VARIATION können direkt auf ASCII-Werte bezogen werden, da als **Elemente** der CypherMatrix alle Zeichen des erweiterten ASCII-Zeichensatzes verwendet werden. Aus den Elementen der BASIC-VARIATION erzeugt das Verfahren in jeder Runde in drei Schleifen die **CypherMatrix** mit 16x16 Elementen.

In Pseudo-Code Darstellung wie folgt:

```

k = Alpha
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
          Matrix$(1,i,j) = CHR$(Variation(k))
          INCR k
          IF k>256 THEN k = 1
        CASE 2
          Matrix$(2,a,j) = Matrix$(1,i,j)
        CASE 3
          Matrix$(3,a,j) = Matrix$(2,i,j)
          CypherSet$ = CypherSet$ + Matrix$(3,a,j)
      END SELECT
    NEXT j
  NEXT i
NEXT s

```

Mit drei Schleifen wird eine vollständige **Permutation** aller Zeichen erreicht. Im String **CypherSet** werden alle Zeichen der Matrix zusammengefasst und als unbegrenzte Byte-Folge in die Datei **SERIES01.RND** geschrieben.

CypherMatrix (16x16) aus der BASIC-VARIATION (256 Elemente)

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

In jeder Runde entsteht eine neue CypherMatrix. Eine Wiederholung derselben Struktur tritt nach den Grundsätzen der Wahrscheinlichkeitsrechnung erst in **256!** (Fakultät) = **8E+506** Fällen auf. Die CypherMatrix ist eine **eindeutige** Abbildung der Eingabe-Sequenz. Sie ist auch kollisionsfrei. Das Ergebnis erfüllt somit alle Eigenschaften einer Hashfunktion und die CypherMatrix in ihrer **Struktur** stellt sich als finaler **Hash-Wert H** dar.

Wenn die Struktur der Matrix einen eindeutigen Hashwert darstellt, dann erfüllen auch alle einzelnen **Zeilen** und **Spalten** die Eigenschaften eines Hashwertes.

Hash-value: Line 7 (CM-final hash)
2C 68 9D 2D 0D 91 32 C4 5C FF D1 3B FC A7 16 DE

Hash-value: Column 12 (CM-final hash)
25 9F 01 E6 D5 F8 3B 06 57 63 CC A1 CB 2B 82 0C

In den Permutationen zur Generierung der CypherMatrix besteht für jedes Zeichen die gleiche Wahrscheinlichkeit an eine andere Stelle der nächsten Matrix umgesetzt zu werden. Infolge Ableitung des Matrix-Schlüssels aus der vorhergehenden Runde und Verwendung als Startsequenz für den folgenden Durchlauf wirken sich alle Permutationen direkt auf die jeweils folgenden Runden aus, und damit auch auf das gesamte Verfahren. Diese Eigenschaft bildet die Grundlage für die Erzeugung unbegrenzter Zeichenfolgen (byte stream).

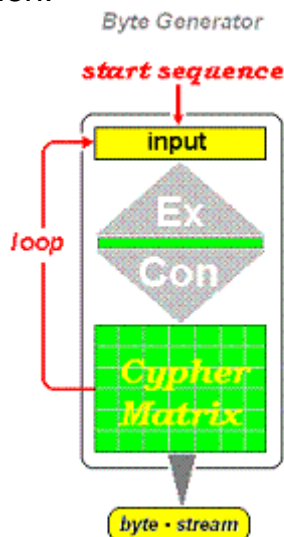
Anwendungen mit dem Daten Generator

Mit Einbindung des Daten Generators zur Lösung kryptographischer Aufgaben sind insbesondere folgende Anwendungen entwickelt:

Generierung von unbegrenzten dynamischen Bytefolgen (Byte Generator),
Berechnung von seriellen und finalen Hash-Werten (CypherMatrix Hash),
einfache und erweiterte Signatur (telePortal) und
alle Arten von Verschlüsselungen mit zahlreichen Operationen.

Byte Generator

Der Byte Generator erzeugt unbegrenzte **Bit-** und **Byte-Folgen** (byte stream). Die Zeichenfolgen können wahlweise in 16-bit, 32-bit, 64-bit oder in anderen Bitfolgen beliebiger Länge geformt werden.



Als Steuerungsparameter werden lediglich **Variante**, **Gamma**, **Delta** und **Theta** eingesetzt. **Alpha** und **Beta** sind nicht erforderlich. Alle Elemente der jeweiligen CypherMatrix mit 256 Bytes werden in der Reihenfolge ihrer Verteilung in jeder Runde zeilenweise in eine Datei (SERIESxx.RND) geschrieben. Die Länge der Folge hängt von der vorzugebenden Anzahl der Runden ab.

Wird die **Kontraktion** anstatt mit MODULO 256 (Verdichtung zur BASIC-VARIATION) anderweitig mit **MODULO 8** (alternativ mit **16**, **32**, **64** oder andere) vorgenommen, lassen sich beliebige Zahlen-Folgen erzeugen, beispielsweise für Verschlüsselungsoperationen. Weitere Einzelheiten im Internet unter:

telecypher.net/CYPHKERN.HTM#Z15

Dynamische Hashfunktion

Als weitere Anwendung bietet sich eine **dynamische Hash-Funktion** an zur eindeutigen Kennzeichnung digitaler Informationen. Dabei werden digitale Folgen (Dateien) in Textblöcke unterteilt, die dann als Eingangs-Sequenzen nacheinander positionsgewichtet, mit der Hashkonstanten **Ck** multipliziert, zur Hash-Funktions-Reihe expandiert und wieder zur GRUND-VARIATION verdichtet einer dreifachen Permutation unterworfen werden. Das Ergebnis - die letzte Cypher Matrix mit 256 Elementen - stellt den **Hashwert (H)** dar. Ein gleicher Hashwert kann erst in **256!** (Fakultät) = **8E+506** Fällen auftreten.

Verglichen mit aktuellen Hash-Funktionen hat das Verfahren einige Vorteile:

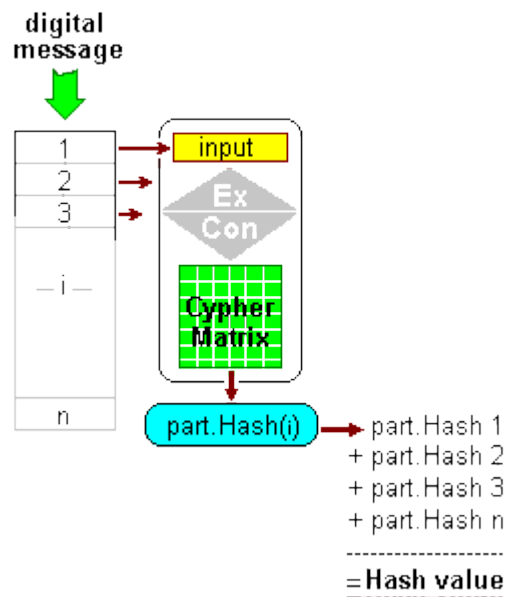
1. Die Einschaltung einer **Kompression** ist nicht erforderlich,
2. Nur Bytes und **einfache** Mathematik werden eingesetzt,
3. Mit den Ergebnissen der Funktion kann **gerechnet** werden (z.B. Addition, Subtraktion, Multiplikation, Division und Modulo-Rechnung) und
4. die Ergebnisse sind beachtlich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

Grundsätzlich sind zwei Techniken möglich:

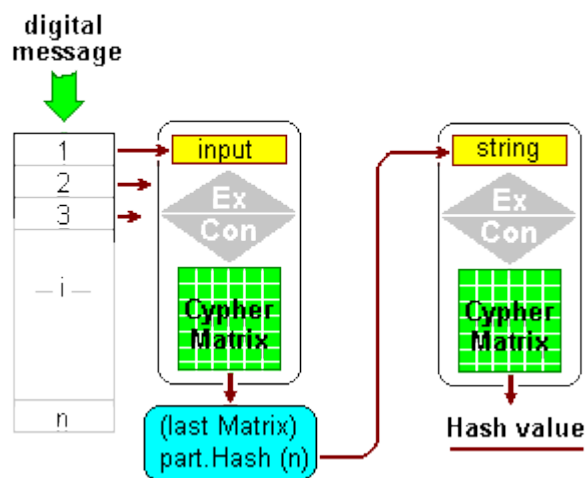
- a) Serieller Abgriff partieller Hashwerte aus der CypherMatrix in jeder Runde und Summierung zum Hashwert (**serial mode**) und
- b) finaler Abgriff des Hashwertes aus der letzten CypherMatrix nach Durchlaufen aller vorherigen Runden (**final mode**).

Die Zusammenhänge veranschaulichen folgende Skizzen:

Serial mode



Final mode



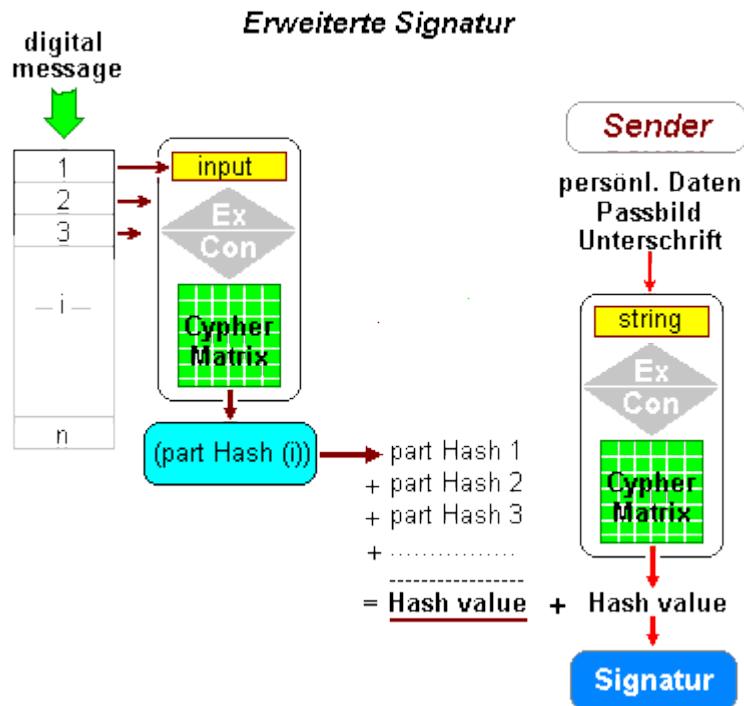
Entwicklung und Arbeitsweise der **CypherMatrix Hash Funktion** werden ausführlich erläutert in den folgenden Pdf-Dateien:

- [Dynamische Hash Funktion](#)
- [Neue Definition der Hash Funktion](#)

Die angesprochenen Hash-Programme stehen an der angegebenen Adresse zum Download bereit.

Erweiterte digitale Signatur

Wird die Hash-Funktion, sowohl auf eine zu **signierende Nachricht** als auch auf die **Identifikations-Daten (ID)** des Signierenden (einschließlich Bild und Unterschrift) angewendet, kann mit Hilfe des CypherMatrix Verfahrens eine **erweiterte digitale Signatur** begründet werden.



Außer den persönlichen Daten des Signierenden kann zusätzlich sein digitalisiertes Photo und seine digitalisierte Unterschrift in die Hashwertberechnung einbezogen werden. Mit dem Programm „**telePort**“ ist der Empfänger der Signatur in der Lage, die **Integrität** der Nachricht und die **Identität** des Absenders anhand der persönlichen Daten, seines Bildes und seiner Unterschrift zu verifizieren. Weitere Einzelheiten im Internet in der pdf-Datei: [Eine einfache digitale Signatur](#)

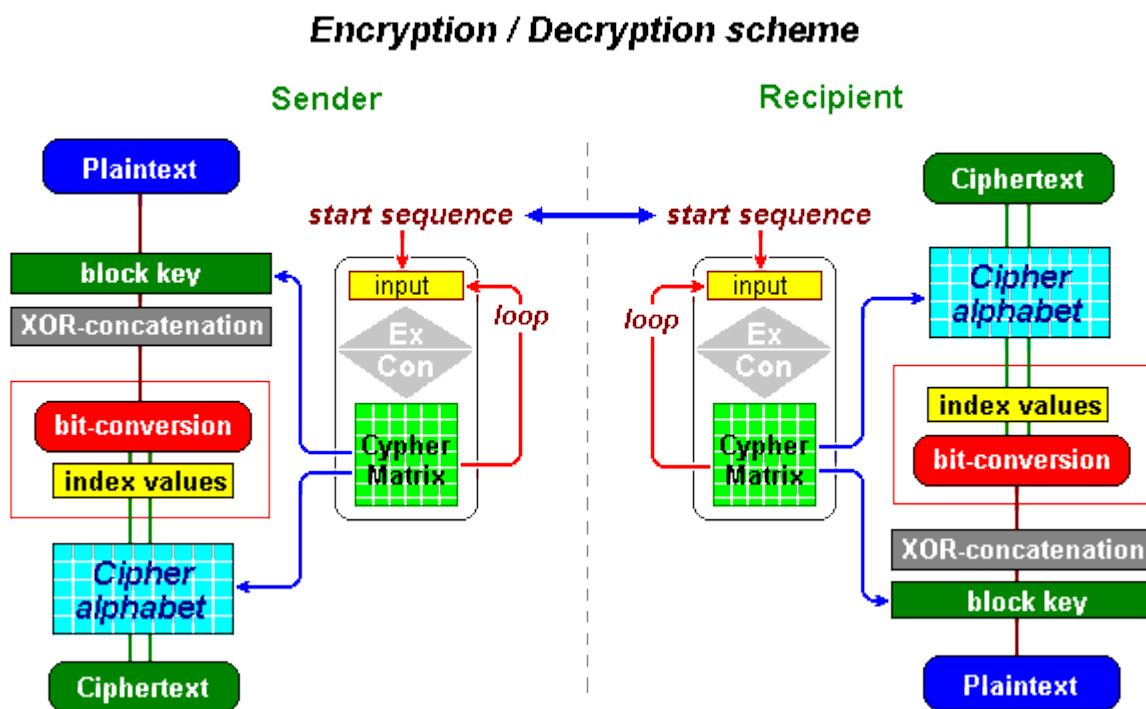
Digitale Verschlüsselungen

Das CypherMatrix Verfahren bewirkt Verschlüsselungen mit einer Vielzahl von Operationen und Kombinationen:

- XOR-Verknüpfung
- Substitution „dyn24“
- Bit-Konversion
- Bit-Exchange
- Structure Changing
- Zahlensystem Basis 4
- Bit Crossing
- Block Transposition
- Zahlensystem Basis 256

Insgesamt sind **52** verschiedene Operationen und Kombinationen praktisch erprobt. Die einzelnen Schritte werden im Internet unter [Digitale Verschlüsselungen](#) im Einzelnen erläutert.

Das Zusammenwirken von **Generator** und **Kodierbereich** zur Durchführung der verschiedenen Operationen und Kombinationen zeigt folgendes Bild:



Als symmetrisches Verfahren geben **Sender** und **Empfänger** eine identische **Startsequenz** ein, die auf beiden Seiten das gesamte Verfahren steuert. Diese Passphrase muss leicht zu behalten sein. Es sollten möglichst einprägsame, vielleicht auch lustige Wortfolgen gewählt werden, die man leicht im Kopf behalten kann und die nicht aufgeschrieben werden dürfen. Hier einige Beispiele:

Im Maschpark laufen die Eisbären an der Leine
 Kangaroos jumping in the Hills of Amarillo
 Blue flamingos flying to Northern Sutherland
 Im Steinhuder Meer wurden 70132 Heringe gezählt

Die Startsequenzen sollten etwa 42 Bytes lang sein, damit wegen ihrer Länge Wörterbuchangriffe und iterative Suche ausgeschlossen sind. Ein Angreifer kann auch nicht mit Erfolg versuchen, Teile der Sequenz getrennt und nacheinander zu analysieren, da die Passphrase nur in einem Durchgang als Ganzes angesprochen werden kann. Die Änderung auch nur eines **einzigsten Bits** sperrt die Funktion und führt zu völlig anderen Ergebnissen.

Verschlüsselungen

Die Verschlüsselung vollzieht sich **in jeder Runde** mit den folgenden 4 Schritten:

1. Generierung der aktuellen **CypherMatrix** (Runden-Matrix) entweder
 - a) mit der eingegebenen **Start-Sequenz** (1.Runde) oder
 - b) mit dem **Matrix-Schlüssel** aus der vorhergehenden Runden-Matrix (ab der 2.Runde)
2. Ableitung des **Chiffre-Alphabets** (128 Zeichen) aus der aktuellen Runden-Matrix ab dem Steuerungsparameter **Alpha**,
3. Ableitung des **Block-Schlüssels** (63 Zeichen) aus der aktuellen Runden-Matrix ab dem Steuerungsparameter **Beta** und
4. Verarbeitung der entnommenen Daten entsprechend der gewählten Operation oder Kombination, beispielsweise der **Bit-Konversion**.

Generierung des Matrix Schlüssels

In der ersten Runde des gewählten Beispiels entnimmt das Verfahren der Runden-Matrix ab Parameter **Gamma** die folgende Zeichensequenz.

Matrix-Schlüssel (ab Offset: **Gamma = 7**)

ë#oK5%|Vâ³¼a?©ï#^0<#PÏÿJ`#ÇÁfz(jÚ#i£¤s#¬!\$^

EB 15 6F 4B 35 25 7C 56 E2 BE 61 3F A9 EF 1F 5E 30 8B 17 50 CF
9F 4A 60 07 C7 C1 66 7A 28 6A DA 14 EC A3 A4 73 01 AC A6 24 88

Der Matrix-Schlüssel wird jeweils auf den Eingang des Verfahrens zurückgeführt (**loop**) und dient zur Initialisierung der nächsten Runde.

Ableitung des Chiffre-Alphabets

Die Bestimmung der Teilmenge „**Alphabet**“ (**128 Zeichen**) geschieht ab dem Parameter **Alpha**, im vorliegenden Fall: Offset = **249**

Ciphertext alphabet: ASCII-Zeichen

Offset: Alpha = 249

Index	1 - 16:	¿ ... ™ k S x ' ö , g µ ë o K 5 %
Index	17 - 32:	V â ¾ a ? © ï ^ 0 < P Ï ÿ J `
Index	33 - 48:	Ç Á f z (j Ú ì £ ¤ s ¬ ¡ \$ ^ †
Index	49 - 64:	È f / ÷ û m b Î » è æ Ö B À d v
Index	65 - 80:	œ T Œ w @ Õ . D x E i t ~ " =
Index	81 - 96:	& [ª ø 4 å Ê , h □ - ` 2 Ä \
Index	97 - 112:	Ñ ; ü \$ % é Å ú " • O Ô " C É _
Index	113 - 128:	y R Ž) ã q ó : ¥ W n M 6 ' * }

Ciphertext alphabet: hexadecimal

BF 85 99 6B 53 D7 92 F5 B8 67 B5 EB 6F 4B 35 25
7C 56 E2 BE 61 3F A9 EF 5E 30 8B 50 CF 9F 4A 60
C7 C1 66 7A 28 6A DA EC A3 A4 73 AC A6 24 88 86
C8 83 2F F7 FB 6D 62 CE BB E8 E6 D6 42 C0 64 76
9C 54 8C AD 77 40 D5 2E 44 78 45 69 74 7E 94 3D
26 5B A0 AA F8 34 E5 CA 2C 68 9D 2D 91 32 C4 5C
D1 3B FC A7 89 E9 C5 FA 93 95 4F D4 22 43 C9 5F
79 52 8E 29 C3 71 F3 3A A5 57 6E 4D 36 27 2A 7D

Bei der Entnahme der Zeichen werden bestimmte Elemente (Steuerungszeichen ASCII-00 bis ASCII-31, ASCII-34, 44 u.a.) ausgeklammert, weil sie in bestimmten Situationen noch ihren eigentlichen Aufgaben nachgehen (z.B. ASCII-26) und den Ablauf durcheinander bringen.

Generierung des Block-Schlüssels

Für XOR-Verknüpfungen mit Klartext-Blocks von wahlweise **63 Zeichen** bestimmt der Parameter **Beta** die Position, ab der eine Sequenz in der gewählten Blocklänge als **Block-Schlüssel** aus der Runden-Matrix entnommen wird.

Block key (at offset: Beta = 93 --> 63 bytes)

4â±Ê,h□-.'2Ä□Ñ;üş#P%œÁú“•O#Ô"#CÉ_yR#Ž#)ÝÃqó:¥WnM6'*}í<G,,!Í€#

34 E5 B1 CA 2C 68 9D 2D 0D 91 32 C4 5C FF D1 3B FC A7 16 DE 89
E9 C5 FA 93 95 4F 00 10 D4 22 06 43 C9 5F 79 52 11 8E 0B 29 DD
C3 71 F3 3A A5 57 6E 4D 36 27 2A 7D ED 3C 47 84 B9 CD 49 80 05

Block-Schlüssel sind nur für Verschlüsselungen erforderlich. Da Klartextblocks und Block-Schlüssel immer gleich lang sind (z.B. 63 Bytes), entsteht ein temporäres „one-time-pad“.

Bit Konversionen

Die **Bit-Konversion** als neuer und wichtigster Verschlüsselungsschritt ist datentechnisch eine Änderung der Anzahl der Bits in den entsprechenden Zeichen. Die Folge der **8-Bit Sequenzen**, beispielsweise aus der XOR-Verknüpfung (0-255), werden in **7-Bit Abschnitte** (0-127) unterteilt, die dann als Indizes (+1) für **128 Zeichen** im **Chiffre-Alphabet** verwendet werden. Kein Bit wird hinzugefügt und kein Bit entfernt. Die Reihenfolge der Ziffern „0“ und „1“ bleibt unverändert, es wird nur eine andere Gruppierung (**8-Bit** --> **7-Bit**) vorgenommen. Als Beispiel:

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

8-bit: 11101111100001111001101010101111101010011101111101000100

7-bit: 11101111100001111001101010101111101010011101111101000100

Eine ausführliche Darstellung der Bit-Konversion finden sich in den pdf-Dateien:

[XOR und Bit Conversion](#)
[Ergänzung zur Bit Conversion](#)

Infolge Bit-Konversion verlängert sich der Chiffretext gegenüber dem Klartext im Verhältnis **7:8**. Damit wird auch die funktionale Verbindung vom **Klartext** zum **Chiffretext** durchbrochen. Das hat entscheidende Auswirkungen auf die Kryptanalyse des Verfahrens. Alle gegenwärtigen Angriffsmethoden – außer „brute force“ - setzen voraus, dass Klartext und Chiffretext gleich lang sind. Es muss also für jedes Klartextzeichen ein bestimmtes Chiffretextzeichen geben. Beim CypherMatrix Verfahren besteht jedoch [keine Längenkongruenz](#) . Auf ein Klartextzeichen entfallen **8/7** Chiffretextzeichen.

Die gesamten heutigen Angriffsszenarien greifen daher nicht. Auch „brute force“ kann bei CypherMatrix Verschlüsselungen mit Bit-Konversion keinen Erfolg haben. Lesen Sie dazu den Nachweis in: [Kryptanalyse des Verfahrens.](#)

München, im September 2008
Ernst Erich Schnoor