

# „Mini-Hash“

## Ein Programm zur Berechnung von Hashwerten

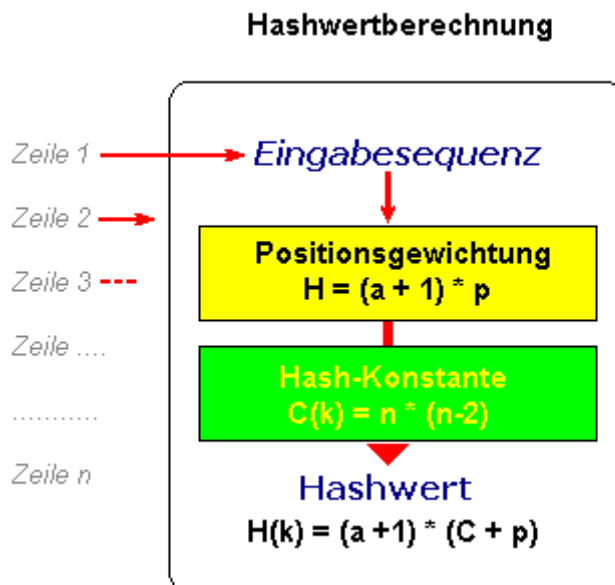
(Ernst Erich Schnoor)

Die wesentliche Aufgabe einer Hashfunktion besteht in der Umformung digitaler Informationen beliebiger Länge ( $n$ ) zu einer eindeutigen digitalen Ausgabe ( $H$ ) in festgelegter Form (**term**). Das kann ein einzelner Wert, eine eindeutige Zeichenfolge (Vektor) oder eine bestimmte Struktur (z.B.: Matrix) sein.

Mit dem “**CypherMatrix**” Verfahren sind bereits eine Reihe von Lösungen realisiert, sichere und kollisionsfreie Hashwerte zu generieren: [>CypherMatrix< als dynamische Hashfunktion](#). Darüber hinaus wird im Folgenden ein Programm vorgestellt, das nur teilweise die Grundsätze des CypherMatrix Verfahrens verwendet, aber einfacher und schneller zu vertretbaren Hashwerten kommt.

### A. Hashwertberechnung

Digitale Zeichenfolgen (Dateien) werden in Klartextblöcke aufgeteilt (z.B. 64 Bytes), die in jeder Runde positionsgewichtet und mit der Hashkonstante **Ck** multipliziert werden, um eine eindeutige Bestimmungsbasis für die Hashwertberechnung zu finden.



Zur Erläuterung der Arbeitsschritte dient die Textdatei: ZAUBER.TXT (1151 Bytes).

*Was ist die Zeit? Ein Geheimnis,- wesenlos und allmächtig. Eine Bedingung der Erscheinungswelt, eine Bewegung, verkoppelt und vermengt dem Dasein der Körper im Raum und ihrer Bewegung. Wäre aber keine Zeit, wenn keine Bewegung wäre? Keine Bewegung, wenn keine Zeit? Frage nur! Ist die Zeit eine Funktion des Raumes? Oder umgekehrt? Oder sind beide identisch? Nur zu gefragt!  
Die Zeit ist tätig, sie hat verbale Beschaffenheit, sie >zeitigt<. Was zeitigt sie denn?*

*Veränderung! Jetzt ist nicht damals, hier nicht dort, denn zwischen beiden liegt Bewegung, an der man die Zeit mißt, kreisläufig, in sich selber geschlossen, so ist das eine Bewegung und Veränderung, die man fast ebensogut als Ruhe und Stillstand bezeichnen könnte; denn das Damals wiederholt sich beständig im Jetzt, das Dort im Hier. Da ferner eine endliche Zeit und ein begrenzter Raum auch mit der verzweifeltsten Anstrengung nicht vorgestellt werden können, so hat man sich entschlossen, Zeit und Raum als ewig und unendlich zu >denken<,- in der Meinung offenbar, dies gelinge, wenn nicht recht gut, so doch etwas besser.*

Thomas Mann, Der Zauberberg, Berlin 1924

Die Eingabe einer Schlüssel-Sequenz zur Initialisierung des Verfahrens ist nicht erforderlich. Die erste Zeile des zu hashenden Textes in der gewählten Länge (64 Zeichen) wird als erste Runde ( $r_1$ ) der Hashfunktion unterworfen. Der Hashwert der Eingabesequenz  $H(r_1)$  wird berechnet.  
Die Eingabesequenz in der ersten Runde ( $r_1$ ) umfasst **64** Bytes und lautet wie folgt:

**Was ist die Zeit? Ein Geheimnis,- wesenlos und allmächtig. Eine**

Das Verfahren durchläuft die folgenden Stufen:

## 1. Verknüpfung der Zeichen

Es gilt eine eindeutige Abbildung der Eingabesequenz als Bestimmungsbasis für die Hashwertberechnung zu finden. Als Einstieg wird eine Verknüpfung der Eingabe durch Addition der Zeichen (Bytes) vorgenommen:

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

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

$$H(k) = \sum_{i=1}^n (a_i + 1)$$
$$H(k) = 5780$$

Der für  $H(k)$  errechnete Wert ist jedoch weit davon entfernt, als Bestimmungsbasis für eine Hashwertberechnung zu dienen. Es müssen weitere Merkmale hinzukommen, um einzelne Bytes ( $a_i$ ) voneinander zu unterscheiden und **Kollisionen** zu vermeiden.

## 2. Positionsgewichtung

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** (analog kartesischem Koordinatensystem) eindeutig bestimmt werden kann. 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 prägnanten Wert für die Folge  $m$  zu erhalten, werden die positionsgewichteten Dimensionswerte zum Wert  $H(k)$  addiert.

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

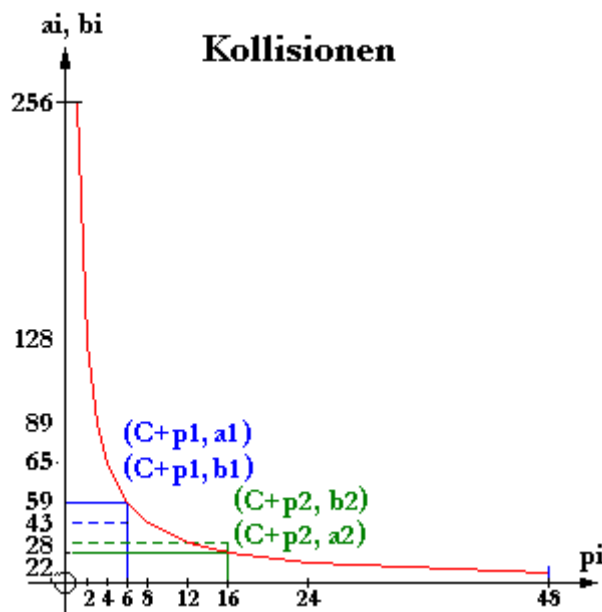
$$H(k) = 188153$$

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.

### 3. Ausschluss von Kollisionen

Die Zusammenhänge zwischen den einzelnen Zeichen  $a_i$  und  $p_i$  im Produkt  $(a_i + 1) * p_i$  zeigt die folgende Kurve:

$a(i), b(i)$ : 1 bis 256  
 $p(i)$ : 1 bis 48



Eine Kollision entsteht, wenn trotz Austausch von Zeichen innerhalb der Eingabesequenz dieselbe Summe  $H(k)$  errechnet wird:

$$H_k(i) = H_k(j)$$

$$(a_i + 1) * p_1 + (a_j + 1) * p_2 = (b_i + 1) * p_1 + (b_j + 1) * p_2$$

Ein Beispiel: In der Eingabesequenz wird an der Position 14 das Zeichen **c** mit dem Zeichen **{** und an der Position 21 das Zeichen **b** mit dem Zeichen **R** vertauscht.

$$a_i + 1 = 99 + 1 = 100 \quad (\text{Zeichen: } c)$$

$$a_j + 1 = 98 + 1 = 99 \quad (\text{Zeichen: } b)$$

$$b_i + 1 = 123 + 1 = 124 \quad (\text{Zeichen: } \{)$$

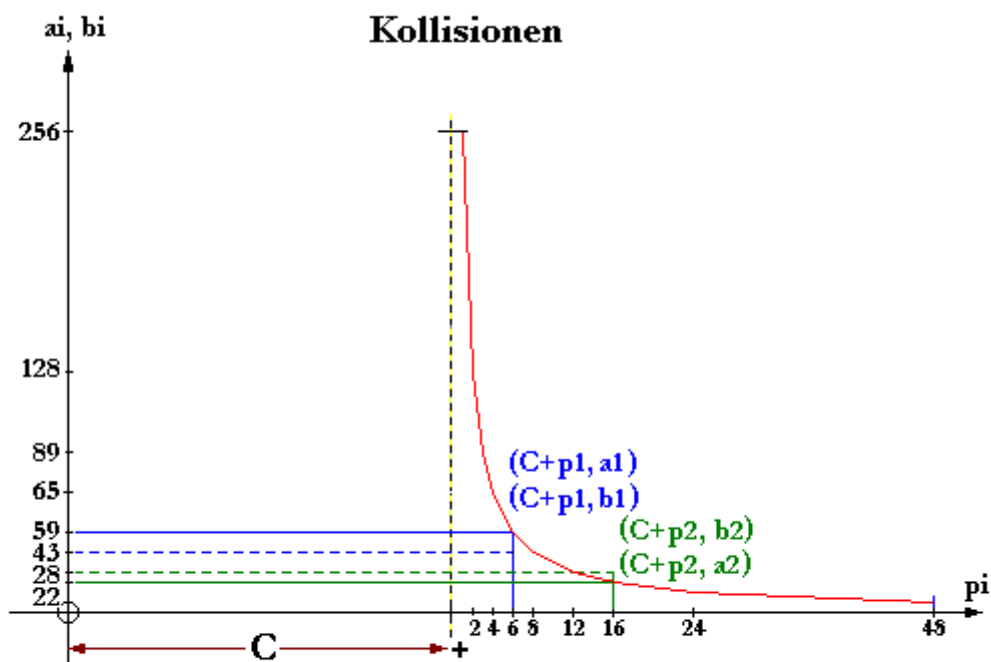
$$b_j + 1 = 82 + 1 = 83 \quad (\text{Zeichen: } R)$$

$$\begin{aligned}
 p_1 &= 14 & p_2 &= 21 \\
 (a_i+1) * p_1 + (a_j+1) * p_2 &= 100 * 14 + 99 * 21 = \mathbf{3479} \\
 (b_i+1) * p_1 + (b_j+1) * p_2 &= 124 * 14 + 83 * 21 = \mathbf{3479}
 \end{aligned}$$

Die Summe der neuen Teilprodukte entspricht der Summe der bisherigen Teilprodukte, so dass an dieser Stelle der Wert  $H(k)$  gleich bleibt, d. h. es entsteht eine **Kollision**.

Um Kollisionen zu vermeiden, wird die Positionsgewichtung in einen Bereich oberhalb der Länge ( $n$ ) verschoben.  $p(i)$  wird um einen konstanten Abstand  $C$  erweitert.

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



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

Nach Umformung ergibt sich der folgende Zusammenhang: Veränderungsquotient  $Q$

$$Q = \frac{(C + p_1)}{(C + p_2)} = \frac{(b_2 - a_2)}{(a_1 - b_1)}$$

Für den „Veränderungsquotienten“ - hier mit  $Q$  bezeichnet – sind drei Fälle möglich:

$$\begin{aligned}
 Q &> 1 \\
 Q &= 1 \\
 Q &< 1
 \end{aligned}$$

Wenn  $Q = 1$ , dann müssen auch  $(C + p_1)$  und  $(C + p_2)$  gleich sein. Da der Austausch an derselben Position  $p$  geschieht, ist hier eine Kollision ausgeschlossen. Ist  $Q > 1$  oder  $Q < 1$ , dann sind auch  $(b_2 - a_2)$  und  $(a_1 - b_1)$  verschieden. Da die Werte  $(a_1, a_2, b_1$  und  $b_2)$  Integerwerte sind, sind auch deren Differenzen ganzzahlig.

Die Positionen  $p_i$  in der Eingabesequenz mit  $N$  Bytes (Länge  $n$ ) umfassen einen Bereich von  $1$  (*minimum*) bis  $N$  (*maximum*), so dass der Veränderungsquotient nach der obigen Formel folgende Spanne erfasst:

$$\frac{C + N}{C + 1} \} Q \{ \frac{N}{N - 1}$$

Durch Auflösung der Zusammenhänge nach  $C$  ergibt sich folgende Berechnung:

$$C = N * (N - 2)$$

Der Faktor  $C$  ist allein von der Länge  $N$  der Eingabesequenz abhängig und hat außerdem die Eigenschaft, für alle Zeichen der Eingabesequenz gleich zu sein: Hashkonstante  $C(k)$ .

Die ausführliche Berechnung ist im Artikel ["Bestimmungsfaktoren für Kollisionsfreiheit"](#) dargelegt. Dem errechneten Wert wird ein individueller Anwender-Code (gewählt aus 1 bis 99) hinzugefügt.

Im vorliegenden Fall wird der Anwender-Code mit  $1$  angesetzt und die Hashkonstante  $C(k)$  errechnet sich mit:

$$C(k) = 64 * (64 - 2) + 1$$

$$C(k) = 3968 + 1 = 3969$$

#### 4. Bestimmungswert $H(s)$ als Summe aller Rundenwerte

Für Hashwertberechnungen digitaler Zeichenfolgen, die länger sind als die im Programm festgelegte Eingabesequenz (z.B. 64 Bytes), durchläuft das Hashprogramm mehr als eine Runde. Insoweit wird zusätzlich die jeweilige Runde ( $r_j$ ) mit ihrer laufenden Nummer im Faktor berücksichtigt (addiert).

Mit Einbindung der Hashkonstanten  $C(k)$  und der laufenden Runden-Nummer ( $r_j$ ) wird der partielle Hashwert der ersten Runde  $H(r_1)$  wie folgt ermittelt:

$$H(r_1) = \sum_{i=1}^n (a_i + 1) * (C_k + p_i + r_1)$$

In Ausschnitten zeigt sich die Berechnung wie folgt:

char	(ai+1)	pi	C(k)	rj	(ai+1)*(C(k)+pi+code+rj)	
					dez	Basis 62
.	...	..	....	.	.....	....
<b>G</b>	72	23	3969	1	287496	1Cn2
<b>e</b>	102	24	3969	1	407388	1hym
<b>h</b>	105	25	3969	1	419475	1i7j
<b>e</b>	102	26	3969	1	407592	1i24
<b>i</b>	106	27	3969	1	423682	1mDa
<b>m</b>	110	28	3969	1	439780	1qPE
<b>n</b>	111	29	3969	1	443889	1rTV
<b>i</b>	106	30	3969	1	424000	1mli
<b>s</b>	116	31	3969	1	460636	1vpc
.	...	..	....	.	.....	....
		Summe	$H(r_1)$		<b>23267875</b>	<b>1Zd2J</b>

Die partiellen Hashwerte der einzelnen Runden werden zum Bestimmungswert **H(s)** summiert. Für die Datei **ZAUBER.TXT** errechnet das Programm den folgenden Bestimmungswert:

DIGIHASH / 64 / 62 / 77 / 1  
 Serielle Rundenwerte in Zauber.txt

dezimal	Basis 62	Runde
23267875	1Zd2J	1
24362632	1eDpg	2
23344581	1ZwzV	3
23400465	1aBWr	4
22364406	1Vq0E	5
23003662	1YWlo	6
23655040	1bFku	7
23551751	1aosx	8
23443581	1aMkH	9
24124205	1dDo5	10
23750908	1behA	11
24284187	1dtQR	12
22887143	1Y1zT	13
24117384	1dC24	14
25234524	1hseS	15
22824146	1XlbO	16
23358361	1a0ZI	17
20291180	1N8f6	18

Hashwert aller Runden  
 dezimal: 421266031  
 Basis 16: 191C026F  
 Basis 62: **SvaXP** **H(s)**

Jede Veränderung in den eingelesenen digitalen Zeichenfolgen führt zu abweichenden Werten im betreffenden Rundenwert und damit auch im Hashwert aller Runden.

Um einen weiteren Sensibilitätsscheck einzuführen, wird der Hashwert aller Runden **H(s)** mit der Anzahl der Runden zu einer weiteren Eingabesequenz zusammengefasst und der Funktion als letzten Durchlauf unterworfen:

Eingabesequenz: SvaXP421266031191C026F18

Das Ergebnis der letzten Runde **H(l)** wird zum Hashwert aller Runden **H(s)** addiert und als **finaler Hashwert H** ausgewiesen.

Hashwert letzte Runde:  
 dezimal: 13377149764  
 Basis 16: 31D571B44  
 Basis 62: EbJ9k4 **H(l)**

**Finaler Hashwert** für: ZAUBER.TXT  
 dezimal: 13798415795  
 Basis 16: 336731DB3  
 Basis 62: **F3okHT** **H**

=====

## B. Sensibilität der Hashfunktion

Zur Darstellung der Sensibilität wird in der Datei **Zauber.txt** die letzte Ziffer der letzten Zeile von ..24 auf ..25 geändert:

(Zauber.txt)

```
..... Der Zauberberg, Berlin 1924
..... 01101001 01101110 00100000 00110001 00111001 00110010 00110100
```

(Zauber1.txt)

```
..... Der Zauberberg, Berlin 1925
..... 01101001 01101110 00100000 00110001 00111001 00110010 00110101
```

char	=	bit
4	=	100
5	=	101

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 1-18:

Hashwert aller Runden  
dezimal: 421270078  
Basis 16: 191C123E  
Basis 62: SVbag

-----  
Hashwert letzte Runde:  
dezimal: 13849672720  
Basis 16: 339813C10  
Basis 62: F7HoY4

### Finaler Hashwert für: ZAUBER1.TXT

dezimal: 14270942798  
Basis 16: 3529D4E4E  
Basis 62: **FZnQ8k**

=====

Der durch Änderung auch nur des letzten Bits von 0 auf 1 verursachte Unterschied in den Hashwerten beträgt:

	Original	Änderung	Differenz
Basis 62:	<b>F3okHT</b>	<b>FZnQ8k</b>	<b>VyfrH</b>
dezimal:	13798415795	14270942798	472527003

## C. Vergleich mit aktuellen Hashberechnungen

Für die Datei **ZAUBER.TXT** errechnen die bekanntesten Hashverfahren die folgenden Hashwerte:

<b>MD 5</b>	22 84 2B 1D E3 E0 34 DB 9C 4F C1 D3 1F 01 B7 28
<b>SHA-1</b>	92 B5 9F 9C 69 38 14 F5 61 5D 58 C7 F3 3E 53 92 C5 2F D8 68
<b>RIPMD-160</b>	E4 BA 42 F2 2F 14 C8 DC 55 D9 A9 BE 97 C9 47 6E 9C 33 D1 FF
<b>DigiHash</b>	F3okHT

Die Hashwerte des **DigiHash**-Programms sind Zahlen, mit denen gerechnet werden kann (alle Grundrechenarten und MODULO-Rechnungen). Im Vergleich zu den im CypherMatrix Verfahren bereits entwickelten Hashwertberechnungen ist das hier vorgestellte Hashprogramm einfacher, schneller und kommt dennoch zu sicheren Hashwerten für alle digitalen Zeichenfolgen.

Die Länge der Hashwerte ist vom Umfang der zu analysierenden digitalen Daten abhängig:  
So zum Beispiel:

Textdatei TAMARA.TXT mit 80.561 Bytes

Finaler Hashwert für: TAMARA.TXT  
dezimal: 76947429000  
Basis 16: 11EA6C9688  
Basis 62: **1LzTZMW**

Datendatei MILLIA.DAT mit 1 MB Zeichen "a"

Finaler Hashwert für: MILLIA.DAT  
dezimal: 1255158577723  
Basis 16: 1243D445743  
Basis 62: **M63pTh5**

Textdatei SIGNATUR.PDF mit 382.878 Bytes

Finaler Hashwert für: SIGNATUR.PDF  
dezimal: 408189830762  
Basis 16: 4496E4E5EA  
Basis 62: **5BYYvjm**

Bilddatei HASHTAB.BMP mit 375.114 Bytes

Finaler Hashwert für: HASHTAB.BMP  
dezimal: 627124642819  
Basis 16: 92038AA403  
Basis 62: **B2X95dr**

Musikdatei ODESOUND.WAV mit 5.727.164 Bytes

Finaler Hashwert für: ODESOUND.WAV  
dezimal: 36413417311080  
Basis 16: 211E28CDCB68  
Basis 62: **AL4sG96m**

Die Länge der Hashwerte bewegen sich etwa zwischen **6** und **8** Ziffern im Zahlensystem zur Basis 62.

Umfang 6 Ziffern:

dezimal: 56.800.235.583  
Basis 16: D388ED03F  
Basis 62: zzzzzz

Umfang 8 Ziffern

dezimal: 218.340.105.584.895  
Basis 16: C694446F00FF  
Basis 62: zzzzzzzz

Für die Hashwerte der Funktion steht somit eine Spanne von Basis 62: **zz000000** Ziffern bzw. dezimal **218.283.305.349.312** Ziffern zur Verfügung.

## D. "Ceterum censeo"

Abschließend einige Gedanken zu den bisherigen Hashwertberechnungen.  
Die traditionellen Hashwerte sind relativ lang:

MD 5:           16 Ziffern im Zahlensystem zur Basis 16  
SHA-1:          20 Ziffern im Zahlensystem zur Basis 16  
RIPEMD-160: 20 Ziffern im Zahlensystem zur Basis 16

Wenn die "Digi-Hash" Funktion auf diese Hashwerte zusätzlich angewendet wird, würden sich wesentlich kürzere Hashwerte ergeben. Zum Beispiel:

Der Hashwert für die Textdatei: **ZAUBER.TXT**

**RIPEMD-160** E4 BA 42 F2 2F 14 C8 DC 55 D9 A9 BE 97 C9 47 6E 9C 33 D1 FF

Die Anwendung der Funktion auf diesen Hashwert führt zu folgendem Ergebnis:

```
DIGIHASH / 64 / 62 / 77 / 1
Serielle Rundenwerte für: RIPEMD.DAT
-----
dezimal   Basis 62   Runde
-----
3748489   Fj9V       1
-----
dezimal:   3748489
Basis 16:  393289
Basis 62:   Fj9V
-----
Hashwert letzte Runde:
dezimal:   3102599757
Basis 16:  B8EDEA4D
Basis 62:   3NyBvZ
-----
Finaler Hashwert für: RIPEMD.DAT
dezimal:   3106348246
Basis 16:  B9271CD6
Basis 62:   3ODv54
=====
```

Das Ergebnis hat nur 6 Ziffern, während das Original 20 Ziffern (40 Zeichen) umfasst. Der neue Hashwert enthält eine eindeutige Abbildung des RIPEMD-160 Wertes. Mit dem Ergebnis kann gerechnet und mit anderen Werten verglichen werden.

Wer die Berechnung von Hashwerten mit dem Programm „**Digi-Hash.exe**“ einmal ausprobieren möchte, kann das Programm und einige weitere Hashprogramme im CypherMatrix Verfahren unter [Demoprogramme zum CypherMatrix Verfahren](#) herunterladen. Zum arbeiten mit den Programmen ist Windows XP erforderlich.

**München, im Juli 2011**