

„Alto-Hash“

Ein Programm zur Berechnung von Hashwerten

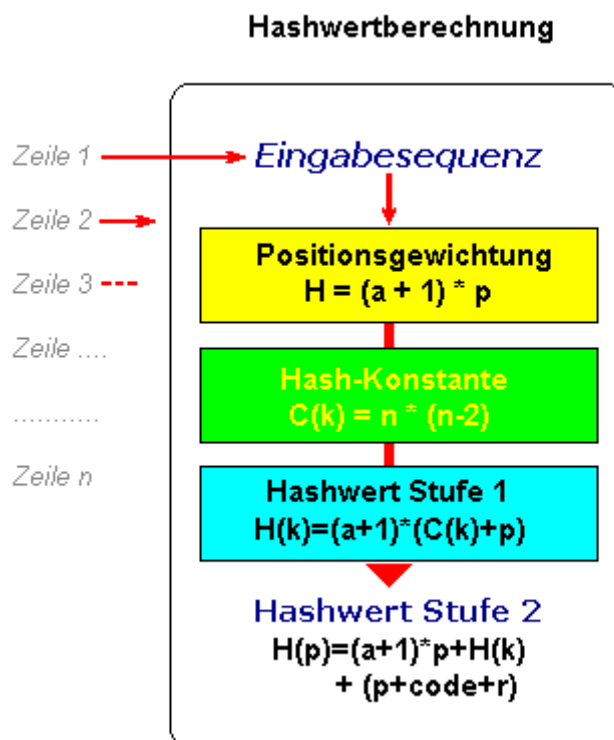
(Ernst Erich Schnoor)

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 weiteres 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 zum Hashwert Stufe 1: **H(k)**. Zur Verdichtung der Bestimmungsbasis werden die Zeichen der Eingangssequenz dann zum Hashwert Stufe 2 : **H(p)** hochgerechnet. Mit einer letzten Runde errechnet das Programm den finalen Hashwert **H** der digitalen Zeichenfolgen.



Zur Erläuterung der Arbeitsschritte dient die Textdatei: MAERCHEN.TXT (1028 Bytes).

Hört zu, ich will euch von einem guten Lande sagen, dahin würde mancher auswandern, wüßte er, wo es läge und fände eine gute Schiffsgelegenheit. Aber der Weg dahin ist weit für die Jungen und die Alten, denen es im Winter zu heiß ist und zu kalt im Sommer.

Diese schöne Gegend heißt Schlaraffenland, auf Welsch Cucagna, da sind die Häuser gedeckt mit Eierfladen, und Türen und Wände sind von Lebzelten, und die Balken von Schweinebraten.

Was man bei uns für einen Dukaten kauft, kostet dort nur einen Pfennig. Um jedes Haus steht ein Zaun, der ist von Bratwürsten geflochten und von bayerischen Würsteln, die sind teils auf dem Rost gebraten, teils frisch gesotten, je nachdem sie einer so oder so gern ißt.

Alle Brunnen sind voll Malvasier und anderer süßer Weine, auch Champagner, die rinnen einem nur so in das Maul hinein, wenn man es an die Röhren hält.

Wer also gern solche Weine trinkt, der eile, daß er in das Schlaraffenland hineinkomme.

Das goldene Märchenbuch, L. Bechstein, Reutlingen

Die Eingabe einer Schlüssel-Sequenz zur Initialisierung des Verfahrens ist nicht erforderlich. Die erste Zeile des zu hashenden Textes in der gewählten Länge 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 **63** Bytes und lautet wie folgt:

Hört zu, ich will euch von einem guten Lande sagen, dahin würde

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)

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

$$H(k) = 5862$$

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.

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

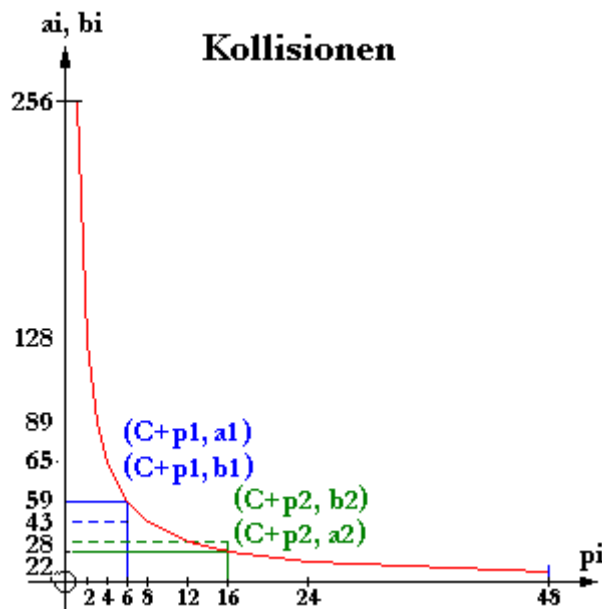
$$H(k) = 188717$$

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

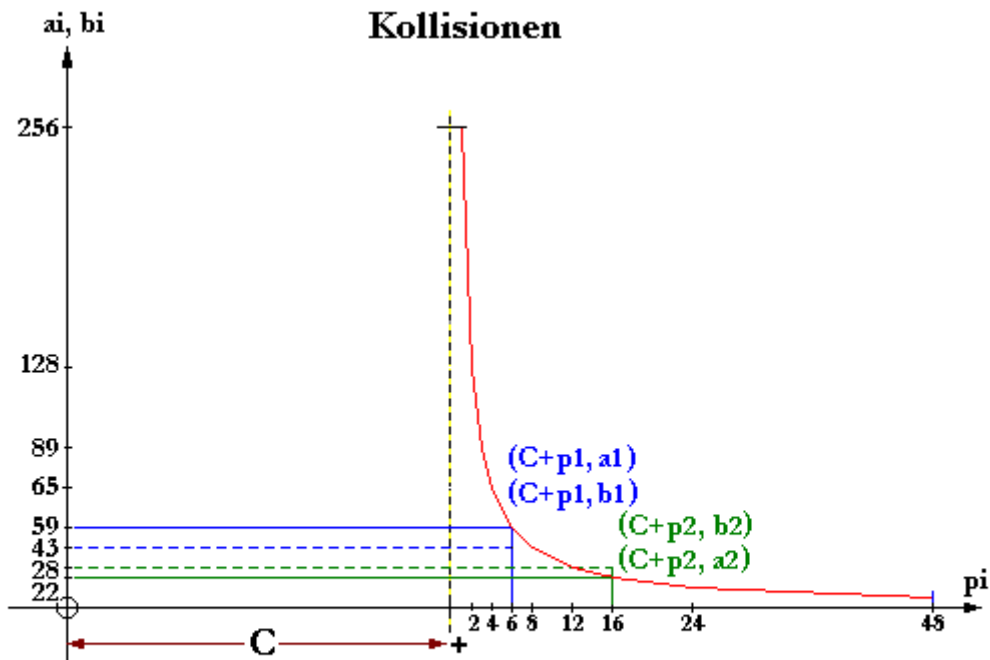
{ und an der Position **21** das Zeichen **b** mit dem Zeichen **R** vertauscht.

$$\begin{aligned}
 a_i + 1 &= 99 + 1 = 100 \quad (\text{Zeichen: } \mathbf{c}) \\
 a_j + 1 &= 98 + 1 = 99 \quad (\text{Zeichen: } \mathbf{b}) \\
 b_i + 1 &= 123 + 1 = 124 \quad (\text{Zeichen: } \mathbf{\{ }) \\
 b_j + 1 &= 82 + 1 = 83 \quad (\text{Zeichen: } \mathbf{R}) \\
 p_1 &= 14 \quad p_2 = 21 \\
 (a_i + 1) * p_1 + (a_j + 1) * p_2 &= 100 * 14 + 99 * 21 = \mathbf{3479} \\
 (b_i + 1) * p_1 + (b_j + 1) * p_2 &= 124 * 14 + 83 * 21 = \mathbf{3479}
 \end{aligned}$$

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:

- $Q > 1$
- $Q = 1$
- $Q < 1$

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.

Die Länge der Eingabesequenz beträgt **64** Bytes. Im vorliegenden Fall ist das letzte Byte jedoch leer „ „, so dass hier das Programm mit der Länge **63** arbeitet. Der Anwender-Code wird mit **1** angesetzt und die Hashkonstante $C(k)$ errechnet sich mit:

$$\begin{aligned} C(k) &= 63 * (63 - 2) + 1 \\ C(k) &= 3843 + 1 = \mathbf{3844} \end{aligned}$$

4. Hashwert Stufe 1

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 Bestimmungswert $H(r_1)$ wie folgt ermittelt:

$$\begin{aligned} H(r_1) &= \sum_{i=1}^n (a_i + 1) * (C_k + p_i + r_1) \\ H(r_1) &= \mathbf{22728107} \end{aligned}$$

Der partielle Bestimmungswert $H(r_j)$ bildet die Basis für Berechnungen von Hashwerten der Stufe1, z.B: Programm **CMH-1 fmx.exe**.

5. Hochrechnung zum Hashwert Stufe 2

Zur Verdichtung der Bestimmungsbasis wird eine **Expansionsfunktion** eingeführt, die die Zeichen auf möglichst viele Variablen erweitert (etwa 160 bis 2400 Zeichen).

Hierzu wird der Wert jedes Zeichens **a(i)** mit der Position **p(i)** und dem bisherigen Zwischenwert **H(r)** verknüpft (multipliziert) und zu einem weiteren Bestimmungswert **H(p)** in den Bereich der **Stufe 2** hochgerechnet.

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

$$H_p = 4289180861$$

Die berechneten Werte umfassen einen Bereich, der für die Berechnung von Hashwerten geeignet ist. Die Werte sind eindeutig und kollisionsfrei. Das Programm **CMH-2 fmx.exe** berechnet Hashwerte der Stufe 2 wie folgt:

CMH-2 fmx / 64 / 62 / 77 / 1

Serielle Rundenwerte in Maerchen.txt

dezimal	Basis 62	Runde
4289180170861	1DVpasyT	1
4640108984896	1JgsyQUK	2
4459829997297	1GW6SANT	3
4423389369225	1FsKJ0MD	4
4714460068464	1L02jznc	5
4423530862628	1FsTshFs	6
4281301037513	1DNEMqSX	7
4370906510551	1Ex2UHJf	8
4579795219557	1ld3C0tZ	9
4585144724974	1litDzd0	10
4520623047577	1HaSfbsv	11
5082948635220	1RUGUT9Y	12
3954775323466	17coUE1a	13
4303965591456	1DlyD0IC	14
4787077695684	1MHJC9vY	15
3607173736118	11VOHOAI	16
2705262	BLIG	17

dezimal: 71024213680749

Basis 16: 40989D06E26D

Basis 62: **KAQ6kWmT** **H(p)**

Um einen weiteren Sensibilitätscheck einzuführen, wird der Bestimmungswert **H(p)** mit der Anzahl der Runden zu einer weiteren Eingabesequenz (**R**) zusammengefasst und der Funktion als letzten Durchlauf unterworfen.

Eingabesequenz (**R**): *KAQ6kWmT7102421368074940989D06E26D17*

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

Hashwert letzte Runde:

dezimal: 105269043816

Basis 16: 1882862268

Basis 62: 1quAB9U **H(I)**

Finaler Hashwert für: MAERCHEN.TXT
 dezimal: 71129482724565
 Basis 16: 40B11F8D04D5
 Basis 62: **KCH0uhvx** **H**
 =====

B. Sensibilität der Hashfunktion

Zur Darstellung der Sensibilität werden in der Datei **Maerchen.txt** die letzten Zeichen der letzten Zeile von ...gen auf ...geo geändert:
 (Maerchen.txt)

..... L. Bechstein, Reutlingen**n**
 01110101 01110100 01101100 01101001 01101110 01100111 01100101 0110111**0**

(Maercheo.txt)

..... L. Bechstein, Reutlinge**o**
 01110101 01110100 01101100 01101001 01101110 01100111 01100101 0110111**1**

char	=	bit
n	=	111 0
o	=	111 1

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 1-17:

Hashwert aller Runden
 dezimal: 71024213705481
 Basis 16: 409 89D074309
 Basis 62: KAQ6kdDN

Hashwert letzte Runde:
 dezimal: 97769987102
 Basis 16: 16C38B9C1E
 Basis 62: 1iietvq

Finaler Hashwert für: MAERCHEO.TXT

dezimal: 71121983692583
 Basis 16: 40AF6092DF27
 Basis 62: **KC8pPX9D**
 =====

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

	Original	Änderung	Differenz
Basis 62:	KCH0uhvx	KC8pPX9D	8BVAmk
dezimal:	71129482724565	71121983692583	7499031982

C. Vergleich mit aktuellen Hashberechnungen

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

MD 5 F0 76 E5 D4 C2 BE 35 7B 65 F7 3A 4E B3 EA D6 0F
SHA-1 1D 18 66 03 8D CB F7 D2 AB 38 BB 12 C7 8A F9 3A 96 88 A1 F8
RIPEMD-160 74 F6 B1 E3 40 0D A4 16 AF 9A D6 37 R8 47 55 9A 15 46 D6 6A

AltoHash **KCH0uhvx**

Die Hashwerte des **AltoHash**-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: 6067869518046864
Basis 16: 158EB1F72AA690
Basis 62: **Rn2FE9J2G**

Datendatei MILLIA.DAT mit 1 MB Zeichen "a"

Finaler Hashwert für: MILLIA.DAT
dezimal: 235128482545537387
Basis 16: 343581BEEB4116B
Basis 62: **HMtEn2doK3**

Textdatei SIGNATUR.PDF mit 382.878 Bytes

Finaler Hashwert für: SIGNATUR.PDF
dezimal: 91356994527299470
Basis 16: 14490B3D68C078E
Basis 62: **6kPn7RHiec**

Bilddatei HASHTAB.BMP mit 375.114 Bytes

Finaler Hashwert für: HASHTAB.BMP
dezimal: 264098789824580767
Basis 16: 3AA4473391ACC9F
Basis 62: **JVzfiORRvz**

Musikdatei ODESOUND.WAV mit 5.727.164 Bytes

Finaler Hashwert für: ODESOUND.WAV
dezimal: 1.08825413776679163E+19
Basis 16: 97068DB98D62E20A
Basis 62: **Cxu9bgX0fSE**

Die Länge der Hashwerte bewegen sich etwa zwischen **7** und **11** Ziffern im Zahlensystem zur Basis 62.

Umfang 7 Ziffern:

dezimal: 3.521.614.606.207
Basis 16: 333F0966F7F
Basis 62: zzzzzzz

Umfang 11 Ziffern

dezimal: 5.20365606838370939E+19
Basis 16: D227003D8D2AF7FC
Basis 62: zzzzzzzzzzz

Für die Hashwerte der Funktion steht somit eine Spanne von Basis 62: **zzzyzzzzzzzw** Ziffern bzw. Dezimal **5.2036557162224877E+19** 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 "Alto-Hash" Funktion auf diese Hashwerte zusätzlich angewendet wird, würden sich wesentlich kürzere Hashwerte ergeben. Zum Beispiel:

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

RIPEMD-160 74 F6 B1 E3 40 0D A4 16 AF 9A D6 37 R8 47 55 9A 15 46 D6 6A

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

```
CMH-2 fmx / 64 / 62 / 77 / 1
Serielle Rundenwerte für: RIPEMD.DAT
-----
      dezimal      Basis 62      Runde
-----
      17221625533   31zE09I      1
-----
dezimal: 17221625533
Basis 16: 2819354BD
Basis 62: 31zE09I
-----
Hashwert letzte Runde:
dezimal: 40199638182
Basis 16: 95C15CCA6
Basis 62: hsXaH0
-----
Finaler Hashwert für: RIPEMD.DAT
dezimal: 212421263715
Basis 16: 31754B4163
Basis 62: 3jrlyQI
=====
```

Das Ergebnis hat nur 7 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 **CypherMatrix** Programmen einmal ausprobieren möchte, kann einige Programme unter [Demoprogramme zum CypherMatrix Verfahren](#) herunterladen. Zum arbeiten mit den Programmen ist Windows XP erforderlich.

München, im Juli 2011