

>CypherMatrix< as dynamic Hash Function

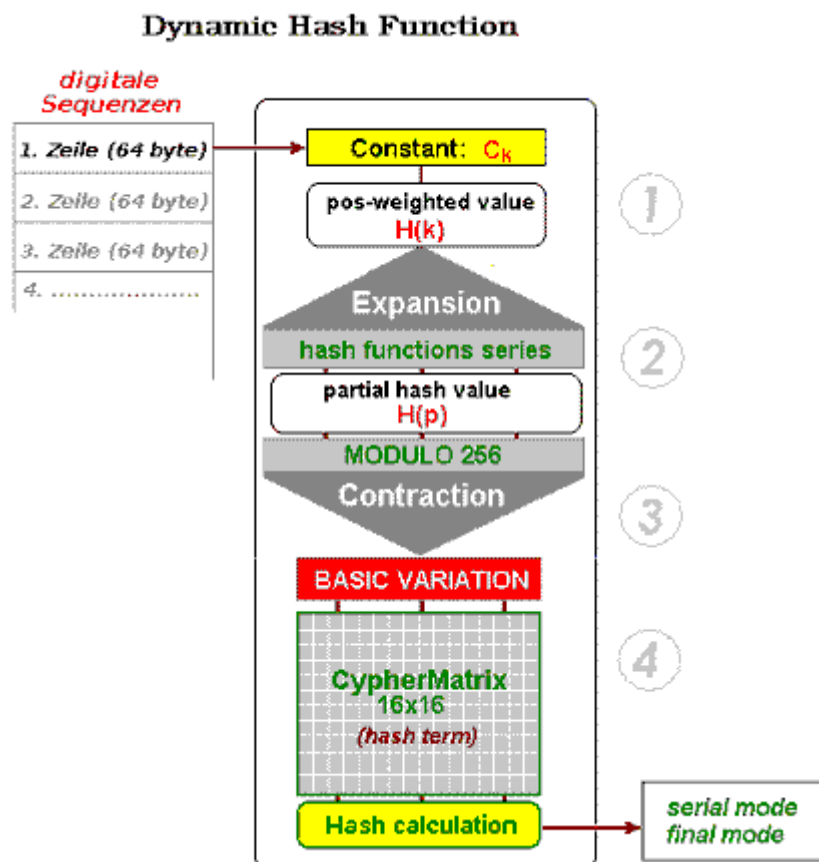
(Ernst Erich Schnoor)

The essential task of a hash function is to perform digital characters of arbitrary length (n) to a unique digital output (H) of specified term [#2]. This can be a single **value**, a unique **string** (vector) or a definite **structure** (e.g. matrix). By means of **Basic Function** a dynamic hash procedure is created, which can be used in various areas of cryptography. You may read a detailed explanation in article: [Cryptographic Basic Function in Byte Techniques](#). Following tract considers last developments of the procedure.

A. Hash Generator

The **Basic Function** forms the core of the hash procedure [#1]. Digital sequences (files) are divided into plaintext blocks (e.g. 64 bytes), which in each cycle will be sequentially position weighted, multiplied with hash constant **C_k**, expanded to hash function series, again contracted to BASIC VARIATION and finally subjected to threefold permutation. The resulting last CypherMatrix with 16x16 elements represents the **HashValue (H)**. A repetition of identic structures due to probability law first occurs in **256!** (faculty) = **8E+506** cases.

The following scheme demonstrates the connections:



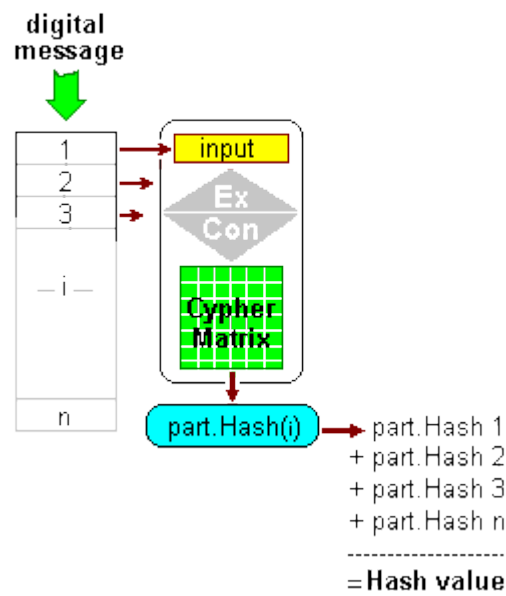
The dynamic hash function works in four steps:

1. Position weighted value $H(k)$ of inserted digital hash sequence with including hash constant $C(k)$ for collision free performances,
2. **Expansion**: extrapolating to a hash function series of about 160 to 2400 digits in number system on **base 77** – first one way function - and calculating an intermediate value $H(p)$,
3. **Contraction**: condensing the hash functions series by Modulo 256 to an array of 16x16 elements: **BASIC-VARIATION** –second one way function - and
4. alternative threefold permutation of BASIC-VARIATION to generate the **CypherMatrix** (16x16 characters) as final result: **CypherMatrix**.

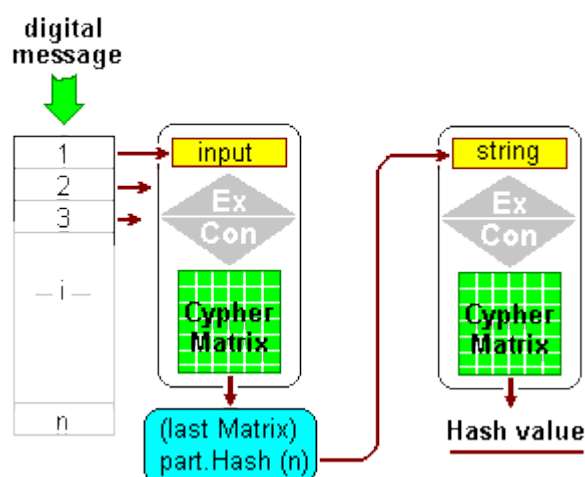
Two techniques of hash processing are possible:

- a) Serial calculation of partial hash values from digital sequences in each round and accumulate to a hash value (**serial mode**) and
- b) final calculation of the hash value from the last CypherMatrix beyond all foregoing partial hash values are added and passed (**final mode**).

Serial mode



Final mode



B. Working steps to CypherMatrix

The text file CANNERY.TXT (401 bytes) serves to explain the working steps:

The WORD is a symbol and a delight which sucks up men and scenes, trees, plants, factories, and Pekinese. Then the Thing becomes the Word and back to Thing again, but warped and woven into a fantastic pattern. The word sucks up Cannery Row, digests it and spews it out, and the Row has taken the shimmer of the green world and the sky-reflecting seas.

John Steinbeck, Cannery Row, New York 1945

Inserting of a separate key sequence to initialize the procedure is not necessary. The first line of the text to be hashed in chosen length (64 characters) will be subjected to the procedure as first round (r_1). Then the hash value $H(r_1)$ of the inserted sequence will be calculated.

The WORD is a symbol and a delight which sucks up men and scenes

The procedure runs the following steps:

1. Position weighting and collision free

First assignment by addition:

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

(Single values for "a_i" are increased by (+1) because otherwise ASCII-zero (0) would not be considered)

$$H(k) = \sum_{i=1}^n (a_i + 1)$$
$$H(k) = 5786$$

But the thus calculated value $H(k)$ is far away to serve as a hash value. To achieve definite results some more features have to be added, especially **position weighting** and **collision free** device.

Each character $a(i)$ is **position weighted** by multiplying its value with its location $p(i)$ [#3]. Time is not relevant.

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

Collisions are avoided by including the **hash constant** $C(k)$. The constant is defined by the length (n) of the input sequence and an individual code (1 to 99).

$$C(k) = n * (n - 2) + \text{code} \quad \text{code} = 1$$
$$C(k) = 64 * 62 + 1 = 3969$$

Derivation of „hash constant“ $C(k)$ you will find in internet at: "[Determinants leading to Collisionfree](#)" [#5]

With inclusion of „hash constant“ $C(k)$ the interim value $H(k)$ will be calculated as follows:

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

$$H_k = 23158479$$

2. Expansion

To obtain more security an **expansion (hash function series)** is introduced which widens the determining factors to a voluminous scale of digits without additional inputs and without losing the quality of being collision free. The number system of expansion can be chosen between 64 ... 77 ... 96. Here **base 77** is fixed. The procedure expands each sequence character to decimal value (s_i) which is changed to (d_i) a digit in number system on base 77 and combined it to **hash function series (i: 1 ... m)**. Simultaneously the procedure calculates the sum of all single results (s_i) as an additional value $H(p)$ in order to fix various destination Data.

$$s_i = (a_i + 1) * p_i * H_k + p_i + \text{code} + \text{cycle}$$

$$s_i \rightarrow d_i \text{ (base 77)}$$

$$H_p = d_1 + d_2 + d_3 + \dots + d_i + \dots + d_m$$

(m = number of digits in number system on base 77)

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

$$H_p = 4489155363963$$

Performing expansion of chosen input sequence (64 bytes) results to the following destination Data:

	decimal	base 62
hash constant $C(k)$:	3969	121
position weighted value (H_k):	23159479	1ZApz
partial serial value (H_p):	4489155363963	1H274Psp
total hash value (H_p+H_k):	4489178523442	1H28daSg

Control parameters derived from destination Data show as follows:

Variante	$(H_k \text{ MOD } 11) + 1$	=	4	begin of reconversion
Alpha	$((H_k + H_p) \text{ MOD } 255) + 1$	=	163	offset initializing
Theta	$(H_k \text{ MOD } 32) + 1$	=	16	dynamic number series

For hash calculation purposes further parameters are not necessary.

By generating hash function series the sequence “ **symbol** “ at position 15 of the input sequence results to the following calculation:

char	pi	Hk	(ai+1)*pi*Hk	Si	base 77			
(ai+1)	(ai+1)*pi		pi+code+r					
s	116	15	1740	23158479	40295753460	17	40295753477	EãMxsu
y	122	16	1952	23158479	45205351008	18	45205351026	Grèzzá
m	110	17	1870	23158479	43306355730	19	43306355749	FëçFcf
b	99	18	1782	23158479	41268409578	20	41268409598	FléIUv
o	112	19	2128	23158479	49281243312	21	49281243333	IFäu1E
l	109	20	2180	23158479	50485484220	22	50485484242	loCeNd
Sum: 4489155363963								LfbETéd

The hash function series **H(p)** results in 383 digits in number system 77 as follows:

tëy**Fy1zQn7**I2ljVçA19èçoz3wá&e&48GãGe4éwMãw4toGàG2foâI095OZAvAâlZ2v3Tâ#ê
 PAaN07@3èRz&G**EãMxsuGrèzzáFëçFcfFléIUvIFäu1EloCeNd**5æfsGNIYR67OL@ãñ2çKã
 âLn74cnâ5Lzk0Fn7lêlrèOF4dèiPNs0tSRêIyZJS8yVævSaYçgUTnse3ZY2pUD09âäd1ba
 èééæIXIUzHâYZkuevXSLruâZæâvâEBiQWdAfqmExUhVs19âbnq9MvfitaC#joNYdTDKyPë
 ymZTâtwlStfäUE90Mv8lë&kcgjTHEhcoPp2eGFléIUâk8âruvrE35hçnJoYpRGSæE4nwgw
 eâjpPv5q0rHëXS&wâzBpUsêTspG#dá7@#

3. Contraction

Next step the hash function series **H(p)** will be contracted by **modulo 256** to the array **BASIC VARIATION** with 16x16 decimal digits. By this the hash function series is assumed to be a series of digits in number system on **base 78** (expansion base 77+1).

First reconversions beginning at **variante = 4** show as follows:

Fy1zQn7				
3 digits	modulo			
base 78	decimal	256	- Theta	element
Fy1	95941	197	16	181
y1z	365179	123	16	107
1zQ	10868	116	16	100
zQn	373201	209	16	193
Qn7	162013	221	16	205
.....

BASIC VARIATION results to the following structure:

BASIC-VARIATION (256 elements)

181 107 100 193 205 168 202 131 247 194 030 166 029 239 187 090
 246 137 209 200 042 178 132 214 120 169 128 052 248 008 101 018
 198 091 080 186 176 199 244 088 089 014 150 245 040 211 180 184
 195 179 114 055 004 001 000 230 027 253 152 145 217 155 085 206
 185 031 005 102 182 201 147 082 207 035 149 075 112 153 086 188
 118 234 119 122 060 229 159 255 167 124 208 087 220 032 203 041

```

151 249 050 092 051 251 108 056 143 170 213 044 010 093 002 009
103 189 115 154 144 003 216 063 068 043 047 057 121 196 177 250
204 218 240 094 125 096 156 025 067 212 252 157 171 197 058 227
215 104 210 172 059 183 097 219 233 254 228 095 221 053 231 222
046 033 161 098 076 006 127 223 175 232 136 020 036 158 162 012
013 123 160 135 224 064 126 037 019 007 099 241 072 016 129 163
011 026 225 034 045 015 061 081 133 105 173 065 038 134 017 174
109 077 235 226 236 028 146 021 130 190 237 022 106 062 023 191
238 024 110 242 071 138 164 243 048 039 139 049 054 192 066 111
069 074 113 116 070 073 117 078 140 079 165 083 084 141 142 148

```

Numbers of **BASIC VARIATION** (16x16) form the base for further hash calculation.

4. Generating CypherMatrix (CM)

Decimal values are related directly to indexes of ASCII-character set and by this to the first **CypherMatrix (CM1)** with 16x16 elements. In order to increase security the elements of BASIC VARIATION are further subjected to a **threefold permutation** and thus form the third **CypherMatrix (CM3)**.

Pseudo code demonstrates as follows:

```

k = Alpha                                     initialising
FOR i = 1 TO 16
  FOR j = 1 TO 16
    Matrix$(1,i,j) = CHR$(Variation(k))       BASIC VARIATION
    INCR k                                     256 elements
    IF k > 256 THEN k = 1
  NEXT j
NEXT i

CM1Set$ = ""                                  elements of first CypherMatrix (CM1)
CM3Set$ = ""                                  elements of third CypherMatrix (CM3)

FOR s = 1 TO 3                                three loops
  FOR i = 1 TO 16                              (permutation)
    FOR j = 1 TO 16
      a = i - j
      IF a <= 0 THEN a = 16 + a
      SELECT CASE s
        CASE 1
          CM1Set$ = CM1Set$ + Matrix$(1,i,j)  CypherString
          Matrix$(2,a,j) = Matrix$(1,i,j)
        CASE 2
          Matrix$(3,i,a) = Matrix$(2,i,j)
        CASE 3
          CM3Set$ = CM3Set$ + Matrix$(3,i,j)  CypherString
      END SELECT
    NEXT j
  NEXT i
NEXT s

```

Derivation first CypherMatrix (CM1) from BASIC VARIATION

```

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

```

The third (final) CypherMatrix (CM3) is deduced from the first CypherMatrix (CM1) by threefold permutation.

Generating third CypherMatrix (CM3) by threefold permutation

```

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

```

The **structure** of CypherMatrix represents a definite mapping of the input sequence. A repetition of identic structures due to probability law first occurs in **256!** (faculty) = **8E+506** cases. But the structure of CypherMatrix as “**Hash Value**” (term) seems to be still very unmanageable. Therefore content of the CypherMatrix has to be transformed to a concise term.

5. Final step: Hash value

Best solution is subjecting the total content of CypherMatrix to the dynamic **hash function**. All elements (e_i) of the CypherMatrix (256 ASCII-characters) fill a **CypherString(CM)** in series with length $n = 256$ which is inserted to the hash function (section B) once more.

$$\text{CypherString(CM)} = e_1 + e_2 + e_3 + \dots + e_i + \dots + e_{256}$$

The hash constant $C(k)$ is calculated as follows:

$$\begin{aligned} C(k) &= n * (n - 2) + \text{code} & \text{code} &= 1 \\ C(k) &= 256 * 254 + 1 = 65025 \end{aligned}$$

In order to avoid collisions the new CypherString is **position weighted**. Further current cycles (r_j) have to be considered because the respective matrix values must differ from each other. To determine hash value $H(r)$ there are two different modes available: (final **step a** and final **step b**):

a) Processing with "step a"

The result of the position weighted **CypherString(CM)** (step 1 of sector B, only) is added by partial serial value $H(p)$ of the foregoing sequence inserted.

$$\begin{aligned} H(\text{CM}) &= \sum_{i=1}^{256} (e_i + 1) * (p_i + C_k + r_j) \\ H(r) &= H(\text{CM}) + H(p) \end{aligned}$$

Derivated from BASIC VARIATION with threefold permutation we get the following hash value $H(r)$:

	decimal	base 62
$H(\text{CM3}) =$	2143304323	2L35MZ
$H(p) =$	4489155363963	1H274Psp

$H(r) =$	4491298668286	1H4S7VFO
=====		

b) Processing with „step b“

The cypherString (CM) of the last CypherMatrix will be subjected to **position weighted** and **expansion** as well (step 1 and step 2 of section B):

$$H(\text{CM}) = \sum_{i=1}^{256} (e_i + 1) * (p_i + C_k + r_j)$$

$$H(\mathbf{CM3}) = \sum_{i=1}^{256} (e_i + 1) * p_i * H(\mathbf{CM}) + p_i + \mathbf{code} + r_j$$

$$H(\mathbf{r}) = H(\mathbf{CM3})$$

Calculation with final “**step b**” results to following final hash value **H(r)**.

	decimal	base 62
H(CM)	2143260811	2L2u2l
H(r)	8998289764804729	fD9w7edF
=====		

Processing mode „**step b**“ is used for all further hash calculations. Instead of partial value of foregoing insertion **H(p)** the new calculated value **H(CM)** from CypherString (CM) is included to the procession. That means more security for the resulting values. Some more alternative solutions for hashing digital series are possible as well.

C. Alternative Hash calculations

If digital character series are longer than chosen input block (e.g. 64 bytes), for instance: longer sequences, text files (messages), drawings, digital images, digital music, programs, measuring results and further digital stored informations, then the hash function procedure has to pass more than one cycle. Basicly two techniques are possible: *serial mode* and *final mode*. The alternatives lead to following variantes:

- A. Program in „**serial mode**“
 - 1. **without** permutation of BASIC VARIATION
 - 2. **threefold** permutation

- B. Program in „**final mode**“
 - 1. on base of **last** CypherMatrix (version: *last cycle LC*)
 - a) **without** permutation
 - b) **threefold** permutation
 - 2. on base of **all** cycles (version: *all cycles AC*)
 - a) **without** permutation
 - b) **threefold** permutation

The variant cases are tabulated in following overview:

Dynamic Hash Functions

Programm	Basis	mode		Permutation		Hashwert
		serial	final	ja	nein	
CMhashAC.exe	AC					VGH3II5eCoC
AChashLT.exe	AC					VSXQJ1EMIoC
CMhashLC.exe	LC					V5IJUJFLLcG
LChashLT.exe	LC					VFpXbQFN0ee
CMhashSE.exe						UFTCGGCX7z6
SEhashLT.exe						UWiAaNQTzfc

meanings: CM = CypherMatrix SE = „serial mode“
 LC = „last cycle“ AC = „all cycles“
 LT = „light“ (without permutation)

1. Hash calculation in „serial mode“

In „serial mode“ the program adds all single hash values $H(r)$ (rounds: $r \rightarrow 1..m$) to a resulting hash value $H(s)$. This term can be established either by **threefold** permutation of BASIC VARIATION or **without** permutation.

$$H(s) = \sum_{r=1}^m H(r)$$

The file **Cannery.txt** has 7 digital input sequences ($m = 7$). Hence, the serial hash value in **serial mode** $H(s)$ comprises the sum of all 7 current single cycle hash values. Dependent on using final „step a“ or final „step b“ there are different courses and results.

a) Course with final „step a“

The file **Cannery.txt** in serial mode with final step a) results in following Data:

```
SMhashSE / 64 / 62 / 77 / 1
Serial Matrix Values in Cannery.txt
-----
```

decimal	base 62	cycle	matrix
4491298668286	1H4S7VFO	1	2L35MZ
4338431727569	1ENajOIN	2	2L3PSI
4303367651135	1DIJk6hT	3	2L30vB
4101815582648	1ADJYATA	4	2L2dcu
4370531275804	1Ewd5pcK	5	2L38Lv
3791932105689	14I3wg93	6	2L2s8h
5967923160	6VsnRA	7	2L2yix

```
-----
```

Serial Hash value for CANNERY.TXT
 decimal: 25403344934291
 base 16: 171AAD59D593
 base 62: **7DEt4UNv**

=====

b) Course with final „step b“

The program **SEhashLT** calculates in serial mode with „step b“ but without permutation for the file „cannery.txt“ the following hash value:

SEhashLT / 64 / 62 / 77 / 1
 Serial Matrix Values in Cannery.txt

decimal	base 62	cycle
9550493406872137	hjxn76i9B	1
8576350492865764	dHLSVoPzQ	2
8938037141205729	ew39gHNNx	3
9084320733581760	fbayIrEGW	4
9354958783796401	gqRIJpDI9	5
9531488216110313	heZC655p3	6
9417597821967145	h8E5dJpqD	7

Serial Hash value for CANNERY.TXT
 decimal: 64453246596399249
 base 16: E4FBE2E3FFC091
 base 62: **4ICBY8HskT**

=====

In „serial mode“ with alternative b) and threefold permutation the definite hash value **H(s)** results as follows:

CMhashSE / 64 / 62 / 77 / 1
 Serial Matrix Values in Cannery.txt

decimal	base 62	cycle	matrix
8998289764804729	fD9w7JedF	1	2L2u2I
8916941149603105	eq3kSgVHN	2	2L2kBI
9123700498778001	fmlrY0nlh	3	2L39Ef
9329230534423440	gj8KmaoG8	4	2L3Y8C
9241077105526001	gK6LMuryL	5	2L3NSB
8678299745665152	dkIKgZWEK	6	2L2HHE
9097764369063568	ffPF6JfcG	7	2L365s

Serial Hash value for CANNERY.TXT
 decimal: 63385303167863996
 base 16: E13098EF4D0CBC
 base 62: **4glvo4PpIM**

=====

2. Hash Values in „final mode“

To calculate the hash value in „final mode“ the serial hash value $H(s)$ is supplemented by a further cycle (round). To achieve this there are two variants to perform the last cycle:

- a) a new input sequence is created out of all elements of the last CypherMatrix and subjected to the dynamic hash function (section B) (version: **last cycle: LC**) or
- b) all serial hash values $H(s)$ of single rounds (in **total** or in **parts**) are combined to a CypherString and as an additional round are subjected to the dynamic hash function (section B). The result added to the serial hash value $H(s)$ represents the final hash value (version: **all cycles AC**).

a) Hash value with elements of last CypherMatrix (version: last cycle)

Despite of chosen block length (e.g. 64 bytes) all 256 elements (e_i) of last Cypher Matrix are aggregated sequently in a new CypherString which forms an additional input sequence subjecting it to the dynamic hash function (section B) and generating a further matrix value $H(I)$.

$$\text{sequence} = e_1 + e_2 + e_3 + \dots + e_i + \dots + e_{256}$$

The thus generated last **CypherMatrix** includes all results of the foregoing rounds. As last step the sum of all rounds $H(s)$ and the additional matrix value $H(I)$ are summerized to form the **definite Hash Value**.

The following list (program: **CMhashLC**) demonstrates the single rounds of file CANNERY.TXT:

```
CMhashLC / 64 / 62 / 77 / 1
Serial Matrix Values in Cannery.txt
-----
```

decimal	base 62	cycle	matrix
8998289764804729	fD9w7JedF	1	2L2u2I
8916941149603105	eq3kSgVHN	2	2L2kBI
9123700498778001	fmlrY0nlh	3	2L39Ef
9329230534423440	gj8KmaoG8	4	2L3Y8C
9241077105526001	gK6LMuryL	5	2L3NSB
8678299745665152	dkIKgZWEK	6	2L2HHE
9097764369063568	ffPF6JfcG	7	2L365s

```
-----
decimal: 63385303167863996
base 16: E13098EF4D0CBC
base 62: 4glvo4PpIM
```

```
-----
Matrix value last cycle:
decimal: 9384220479087873
base 16: 2156E618266501
base 62: gykSlkb7x
-----
```

Final Hash value for: CANNERY.TXT
 decimal: 72769523646951869
 base 16: 102877F077371BD
 base 62: **5NHgGqAQtJ**

=====

b) Aggregating all serial hash values (version: all cycles)

The program takes the respective last three digits (number system on base 62) from each serial hash value **H(r)** and then combines them to a new sequence to be inserted.

The extracts are limited to three digits because otherwise in case of voluminous files the sequences and thus the hash values as well would become too long. The sum **H(s)** of serial hash values is set in front of the the sequence to be inserted.

Concerning the **Cannery.txt** the program **CMhashAC.exe** calculates the following hash values:

CMhashAC / 64 / 62 / 0 / 1
 Serial Matrix Values in Cannery.txt

decimal	base 62	cycle
8998289764804729	fD9w7J edF	1
8916941149603105	eq3kSg VHN	2
9123700498778001	fmlrY0 nlh	3
9329230534423440	gj8Kma oG8	4
9241077105526001	gK6LMury L	5
8678299745665152	dklKg ZWEK	6
9097764369063568	ffPF6J fcG	7

decimal: 63385303167863996
 base 16: E13098EF4D0CBC
 base 62: **4glvo4PplM** **H(s)**

The sequence to be inserted to the last processing cycle reads as follows:

Sequence = **4glvo4PplMedFVHnlhoG8ryLWEKfcG**

The program calculates this sequence as last cycle to interim matrix value **H(l)** which is added to the sum **H(s)** in order to form the **final hash value H**.

Matrix value last cycle:
 decimal: 9161800959655652
 base 16: 208C9C03B3FEE4
 base 62: **fxaduFpMS** **H(l)**

Final Hash value for: CANNERY.TXT
 decimal: 72547104127519648
 base 16: 101BD34F3010BA0
 base 62: **5MGWRyff7o** **H**

Of all here demonstrated examples the program „**CMhashLC**“ seems to be best qualified for practical using. Lengths of hash values are from **9** to **12** digits in number system on base 62. Thus the range of the hash values stretch from 62^9 to 62^{12} , resp. in decimal numbers:

1.35370865462636E+16 up to **3.2262667623979E+21**

D. Sensitivity analysis

In order to demonstrate the sensitivity of the hash function the last character in the last line of file **Cannary.txt** is changed from ..**5** to ..**4**:

```

                                John Steinbeck, Cannery Row, New York 1945
..... 01110010 01101011 00100000 00110001 00111001 00110100 00110101
                                John Steinbeck, Cannery Row, New York 1944
..... 01110010 01101011 00100000 00110001 00111001 00110100 00110100

char      bit
5      =  00110101
4      =  00110100

```

The last bit „**1**“ is set to „**0**“. The entire rest remains unchanged.

Calculation of the respective hash value by the program **CMhashLC.exe** leads to the following Data:

Primary text file

```

CMhashLC / 64 / 62 / 77 / 1
Serial Matrix Values in Cannery.txt
-----
decimal: 63385303167863996
base 16: E13098EF4D0CBC
base 62:      4glvo4PpIM
-----
Matrix value last cycle:
decimal:  9384220479087873
base 16:  2156E618266501
base 62:      gykSlkb7x
-----
Final Hash value for: CANNERY.TXT
decimal: 72769523646951869
base 16: 102877F077371BD
base 62:      5NHgGqAQtJ
=====

```

Changed in one „bit“

```
CMhashLC / 64 / 62 / 77 / 1
Serial Matrix Values in Cannery.chd
-----
decimal: 63761782556493548
base 16:  E28700DF8686EC
base 62:  4i1pwB3Wfo
-----
Matrix value last cycle:
decimal: 8899444335576713
base 16:  1F9DFF5A520689
base 62:  el5huDwsL
-----
Final Hash value for: CANNERY.CHD
decimal: 72661226892070261
base 16:  102250039D88D75
base 62:  5Mmve5HT89
=====
```

The change caused by one bit results in a hash value difference as follows:

	original	changed	difference
base 62	5NHgGqAQtJ	5Mmve5HT89	UkcksxIA
decimal	72769523646951869	72661226892070261	108296754881608

E. CypherMatrix compared with actual processes

In researching process of digital facts many hash functions have been developed, which fulfill their task more or less well. As problem remains however that procedures must be safe enough (clear and undisputable). The so far used procedures of the SHA family are lately endangered by special attacks. Thus NIST [#9] at present accomplishes an international competition, in which at the end the best suitable procedure is to be elected.

Those candidates already became known [#10] are essentially arranged according to a pattern, which corresponds to the structure of previous hash functions to a large extent. In order to compare with CypherMatrix hash values a confrontation of the two ranges is shown in the following tabula.

Accordingly – strictly speaking – CypherMatrix is a pure **mathematical** procedure. There are only few connections to current hash algorithms.

Hash Functions (general)	CypherMatrix Hash Function
Fundamental Elements	
bits	bytes
Additional Functions	
IVs, SALT, padding	none
Working Steps, Sequences (message digest)	
224, 256, 384, 512, 1024 bits (in part: variable)	continuous, unlimited (optimal: 16 up to 256 bytes)
Internal Consistence (internal states)	
Feistel network keys, S-boxes, constants, shifting, rotation, mixing, XOR swapping, permutations	position weighted hash constant C expansion, contraction (one way functions)
Compression Function	
„Merkle-Damgård“ block ciphers, AES-based Threefish based	none
Output Function	
output function truncated to output fixed length	CypherMatrix: $GF(16^2)$ threefold permutation number system to basis 62 (9 up to 11 digits)
Applications (Anwendungen)	
hashing, MAC, randomizing, PRNG, digital signatures, authentication, encryptions	hash funktion, randomizing digital signatures, RNG, authentication, encryptions

F. References

- [#1] Author's patent, DPMA 19811593 from 18.03.1998
- [#2] Knudsen, Lars R., Lai, Xuejia and Preneel, Bart, Attacks on Fast Double Block Length Hash Functions, Journal of Cryptology, Vol.11#1, New York, 1998, p.59
- [#3] analogous to: Descartes, René, (without more indication)
- [#5] Collisionfree: telecypher.net/Collfree.pdf
- [#6] Article at: telecypher.net/CORECYPH.HTM#Z15
- [#9] NIST: National Institute of Standards and Technology, USA
- [#10] [Grøstl, LANE, SHAvite-3, Skein, TIB3, ...](#)
siehe: [//ehash.iak.tugraz.at/wiki/The_SHA-3_Zoo](http://ehash.iak.tugraz.at/wiki/The_SHA-3_Zoo)

Munich, in June 2010