

## >CypherMatrix< as dynamic Hash Function

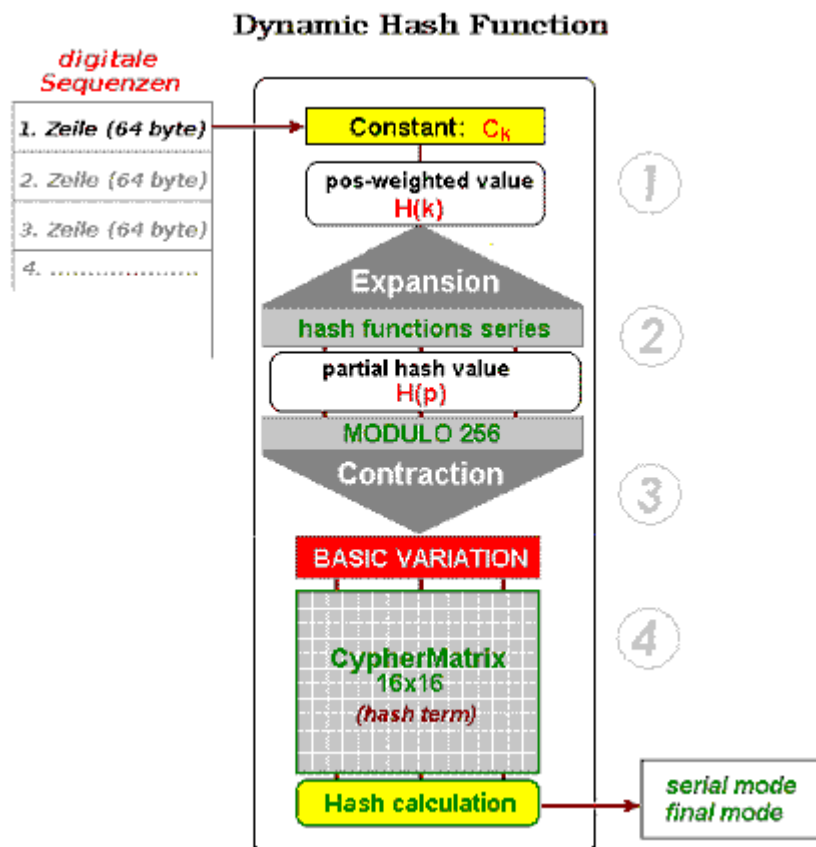
(Ernst Erich Schnoor)

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. Superseded solutions are not demonstrated any longer.

### 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:



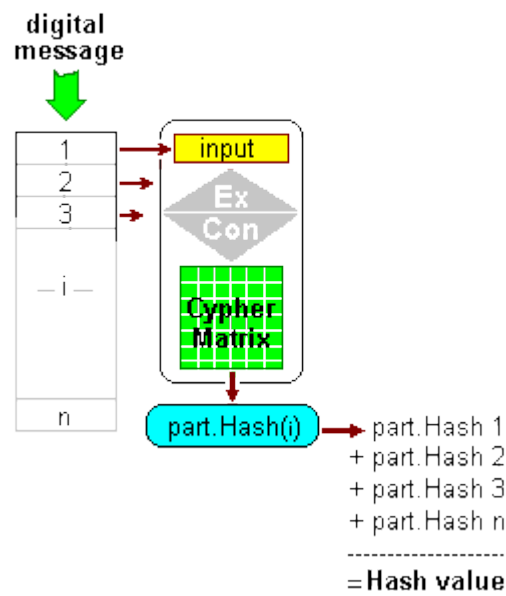
The dynamic hash function works in four steps:

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**.

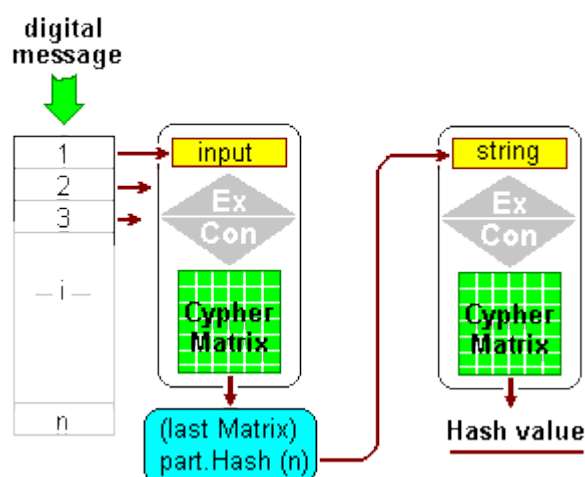
Two techniques of hash processing are possible:

- 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

A definite mapping of the inserted sequence as determination base for calculating hash values has to be searched for. As an entry into searching an addition of characters (bytes) is performed.

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

(Single values for "a<sub>i</sub>" 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 determination base for calculating hash values. 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 condense the determination base 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 up to 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)

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

$$H_p = 4489155363963$$

Number system on base 77 comprises the following digits:

0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ  
 abcdefghijklmnopqrstuvwxyz&#@àáâãäåæçèéë  
 (defined by the author, not standardized)

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

	decimal	base 77
hash constant $C(k)$ :	<b>3969</b>	<b>pg</b>
position weighted value ( $H_k$ ):	<b>23159479</b>	<b>ouAZ</b>
partial serial value ( $H_p$ ):	<b>4489155363963</b>	<b>LfbETéd</b>

Only **Variante**, **Alpha** and **Theta** are necessary for further researching. The other parameters are not needed.

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

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

char	pi	Hk	$(a_{i+1}) * pi * Hk$	Si	base 77			
	$(a_{i+1})$	$(a_{i+1}) * pi$		$pi + code + r$				
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								<b>LfbETéd</b>

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

tëyFy1zQn7I2ljVçA19èçoz3wá&e&48GãGe4éwMāw4toGàG2foâI095OZAvAâlZ2v3Tâ#ê  
 PAäN07@3èRz&GEãMxsuGrèzzáFëçFcfFléIUvIFäu1EloCeNd5æfsGNIYR67OL@ãñ2çKä  
 âLn74cnâ5Lzk0Fn7lêlrèOF4dèiPNs0tSRêIyZJS8yVævSaYçgUTnse3ZY2pUDO9âäd1ba  
 èèéæIXIUzHâYZkuevXSLruâZæâvâEBiQWdAfqmExUhVs19âbnq9MvfitaC#joNYdTDKyPë  
 ymZTâtwlStfäUE90Mv8lë&kcgjTHEhcoPp2eGFléIUâk8âruvrE35hçnJoYpRGSæE4nwgw  
 eâjpPv5q0rHëXS&wâzBpUsêTspG#dá7@#

### 3. Contraction

In order to lead back the determination base to decimal dimensions next step the hash function series **H(p)** will be contracted by **modulo 256** to 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:

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

**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.

#### 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 elements of the first CypherMatrix (**CM1**) can be used directly as determination base for calculating hash values. But to increase security an additional threefold permutation is introduced which results in **CypherMatrix (CM3)**. To perform the permutation three variants (A, B and C) are possible.

Pseudo code demonstrates as follows:

### a) simply replacing index: i (variant A)

```
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,a,j) = Matrix$(2,i,j)
        CASE 3
          CM3Set$ = CM3Set$ + Matrix$(3,i,j)  CypherString
      END SELECT
    NEXT j
  NEXT i
NEXT s
```

### b) displaced changing of indexes: i,j (variant B)

```
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
```

### c) dynamic generating of indexes: i,j (variant C)

All index values (16x16) are generated anew in a separate index-array **Index\$(16,16)** (application of CypherMatrix function):

```
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$(1,i,j)
  CASE 2
    x = VAL(Index$(i,j))
    Matrix$(3,i,x) = Matrix$(2,i,j)
  CASE 3
    CM3Set$ = CM3Set$ + Matrix$(3,i,j)  CypherString
END SELECT
```

The third (final) CypherMatrix (CM3) is deduced from first CypherMatrix (CM1) as follows:.

### Generating third CypherMatrix (CM3) by threefold permutation (variant B)

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 and by this is qualified as determination base to calculate hash values. 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. Hash value calculating H(r)

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.

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

In our case the following CypherString is subjected to the hash function:

```
{.ã±]ÜK~#xfuŠi" á#çÄy,Ð##X,,Fítn#m£ç5«9Ö|
íæô²Ç*ÄqJi®□žÝ□/ª§R#“#°ÈdkE¿##$ ü+□ÿ8ÿÉ#°Ñ%µo#†H#äÔDC?
lâ¶7P[ö”B>&ñ^pèé#Øû<fr³ÆZZÀjAc~# Üœ#3z##Ã#»□6#1ii#ßa`□\wê¹,eiT#S<¾4...%□·}š2ùvî
'Öø!¥',Q~#,^s½—¼UV>(4#O0#=@L-ðÜg)Ë™Üö€Âœó'ábÒhì×ú# p'~©÷Nα#-‡j!
```

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

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

$$C(k) = 256 * 254 + 1 = 65025$$

In order to avoid collisions the new CypherString is **position weighted**.

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):

$$\mathbf{CM(k)} = \sum_{i=1}^{256} (\mathbf{e_i + 1}) * (\mathbf{p_i + C_k})$$

$$\mathbf{CM(p)} = \sum_{i=1}^{256} (\mathbf{e_i + 1}) * \mathbf{p_i} * \mathbf{CM(k)} + \mathbf{p_i + code}$$

$$\mathbf{H(r)} = \mathbf{CM(p)}$$

For better handling the hash value is shown in digits of number system on bas 62 which comprises the following numbers:

0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ  
 abcdefghijklmnopqrstuvwxyz  
 (defined by the author, not standardized)

The final hash value  $\mathbf{H(r_1)}$  of first start sequence results as follows:

	decimal	base 62
$\mathbf{CM(k)}$	$\mathbf{= 2143377944}$	$\mathbf{2L3OW0}$
$\mathbf{H(r_1)}$	$\mathbf{= 9179333265149120}$	$\mathbf{g2ZJCcxeq}$
	=====	

In further hash calculations **variant B** (displaced changing of indexes: i,j) will be used. 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			B		TGZRB0WG8S0
AChashLT.exe	AC					VZC6D39V6no
CMhashLC.exe	LC			B		TLTh3SVvW7Y
LChashLT.exe	LC					URo4Lo6yvCC
CMhashSE.exe				B		TCnaQ0uu7qy
SEhashLT.exe						T9HM60CfiCG

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 **without** or **with threefold permutation** there are different courses and results.

Program **SEhashLT** calculates in serial mode without permutation the following hash value for the file „Cannery.txt“:

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

---

decimal	base 62	cycle	CM(k)
9177625875610505	g25FViSPR	1	2L3OJB
9047399075016964	fR6X7yZng	2	2L3H4g
9004700087069993	fEynGqbUH	3	2L3KRX
9248631284063376	gMFL5oAMq	4	2L3wWy
9017297474745232	fIYZspUPY	5	2L3d3E

```

8930014319381028  etluPabae  6  2L3b1G
8252507714858112  bnO1cchsm  7  2L2PTs

```

```

-----
Serial Hash value for: CANNERY.TXT
decimal: 62678175830745210
base 16:  DEAD78058FA07A
base 62:   4d48QvObxC          H(s)
=====

```

In „serial mode“ with 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  CM(k)
-----
9179333265149120  g2ZJCcxeq    1  2L3OW0
8875591246015012  eeJlBt11A    2  2L2wGI
8876597508913628  eebTZVad2    3  2Jm2hN
8633026986588816  dXRHRUBEW    4  2L2jxU
9149020339451776  ftxdHrvO4    5  2L3t0S
9292917235850244  gYp1BwgSK    6  2L4lyY
9215450950314884  gCpBGzAG4    7  2L4l8M
-----

```

```

-----
Serial Hash value for: CANNERY.TXT
decimal: 63221937532283480
base 16:  E09C04689FAA58
base 62:   4fYXfAJWe0          H(s)
=====

```

## 2. Hash calculation 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 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 additional **Matrix value H(I)** and the sum of all foregoing rounds **H(s)** are summerized. The result forms the **definite Hash Value H**

The following list (program: **CmhashLC** realized as **DynaHash.exe**) demonstrates the single rounds of file CANNERY.TXT:

```

DynaHash / 64 / 62 / 77 / 1
Serial Matrix Values in Cannery.txt
-----
      decimal          base 62    cycle  CM(k)
-----
9179333265149120    g2ZJCcxeq      1  2L3OW0
8875591246015012    eeJIBt11A      2  2L2wGI
8876597508913628    eebTZVad2      3  2Jm2hN
8633026986588816    dXRHRUBEW      4  2L2jxU
9149020339451776    ftxdHrvO4      5  2L3t0S
9292917235850244    gYp1BwgSK      6  2L4IyY
9215450950314884    gCpBGzAG4      7  2L4I8M
-----
decimal: 63221937532283480
base 16:  E09C04689FAA58
base 62:  4fYXfAJWe0          H(s)
-----
Matrix value last cycle:
decimal:  9214460994866052
base 16:  20BC80E2654B84
base 62:  gCXkgyzHo          H(I)
-----
Final Hash value for: CANNERY.TXT
decimal: 72436398527149532
base 16: 10158854B04F5DC
base 62:  5LI5PriVvo          H
=====

```

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

The sum of all hash values **H(s)** is supplemented by an additional round. The program takes the respective last three digits (number system on base 62) from each serial hash value **H(r)** and 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. By this all results of the foregoing rounds are included into the final hash calculating. The result of the additional round **H(I)** and the sum **H(s)** of serial hash values are added to form the **final hash value H**.

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

```

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

```

decimal	base 62	cycle	CM(k)
9179333265149120	g2ZJCc <b>xeq</b>	1	2L3OW0
8875591246015012	eeJIBt <b>11A</b>	2	2L2wGI
8876597508913628	eebTZV <b>ad2</b>	3	2Jm2hN
8633026986588816	dXRHRU <b>BEW</b>	4	2L2jxU
9149020339451776	ftxdHrv <b>O4</b>	5	2L3t0S
9292917235850244	gYp1Bwg <b>SK</b>	6	2L4lyY
9215450950314884	gCpBGz <b>AG4</b>	7	2L4I8M

decimal: 63221937532283480  
base 16: E09C04689FAA58  
base 62: 4fYXfAJWe0 **H(s)**

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

Sequence = **xeq11Aad2BEWvO4gSKAG4**

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

Matrix value last cycle:  
decimal: 9214460994866052  
base 16: 20BC80E2654B84  
base 62: gCXkgyzHo **H(I)**

**Final Hash value** for: CANNERY.TXT  
decimal: 72436398527149532  
base 16: 10158854B04F5DC  
base 62: **5LI5PriVvo** **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 stretches 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 **..45** to **..44**:

```

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 following Data:

### Primary text file

```
DynaHash / 64 / 62 / 77 / 1
Serial Matrix Values in Cannery.txt
-----
decimal: 63221937532283480
base 16: E09C04689FAA58
base 62: 4fYXfAJWe0 H(s)
-----
Matrix value last cycle:
decimal: 9214460994866052
base 16: 20BC80E2654B84
base 62: gCXkgyzHo H(I)
-----
Final Hash value for: CANNERY.TXT
decimal: 72436398527149532
base 16: 10158854B04F5DC
base 62: 5LI5PriVvo H
=====
```

### Changed last „bit“ (file: Cannery.chd)

```
DynaHash / 64 / 62 / 77 / 1
Serial Matrix Values in Cannery.chd
-----
decimal: 63042067281734244
base 16: DFF86D1A3BEA64
base 62: 4ejSwjanFE H(s)
-----
Matrix value last cycle:
decimal: 9034610028675728
base 16: 2018EE11718290
base 62: fNTNJLSa0 H(I)
-----
Final Hash value for: CANNERY.CHD
decimal: 72076677310409972
base 16: 100115B2BAD6CF4
base 62: 5K6wK2wFpE H
=====
```

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

	original	changed	difference
base 62	<b>5LI5PrIVvo</b>	<b>5K6wK2wFpE</b>	<b>1e95oMG6a</b>
decimal	72436398527149532	72076677310409972	359721216739560

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

## F. Concluding remarks

Detailed explanations and further examples to CypherMatrix procedure are presented in internet at: <http://www.telecypher.net/>.

In section C. Dynamic Hash Functions shown programs may be downloaded from <http://www.telecypher.net/DELIVERY.HTM> and tested in detail.

## **G. 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) Cartesian coordinate system:  
object = subject \* location \* time
- [#5] Collisionfree: [telecypher.net/Collfree.pdf](http://telecypher.net/Collfree.pdf)
- [#6] Article at: [telecypher.net/CORECYPH.HTM#Z15](http://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 July 2011