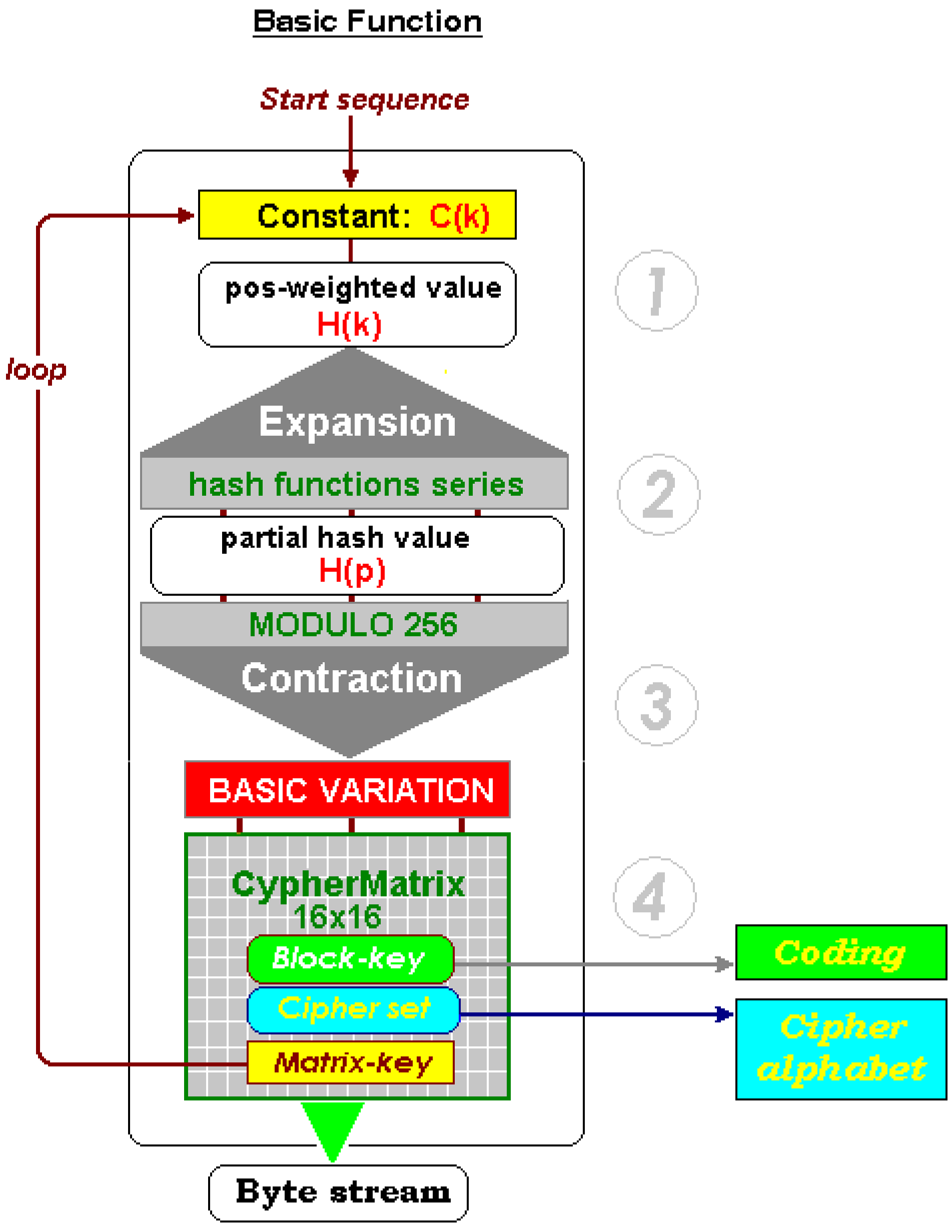
Cryptographic Basic Function in Byte Techniques

(Ernst Erich Schnoor)

I. Basic Function

First a fundamental function is explained which begins with an input of a short message (**start sequence**) and processes this information to a definite hash term (named by the Author :"**CypherMatrix**" procedure: DPMA 19811593 as of 18.03.1998). The function uses **bytes** only, simple mathematics including MODULO calculation and number systems from base 2 up to base 256. But, no manipulations of **bits,** at all. The following scheme demonstrates the connections:



Second is demonstrated how this function can be combined with other devices to solve cryptographic problems, especially encryption, hash calculation, signatures and random series.

The basic function begins with a **start sequence** as input (e.g. 42 bytes). The start sequence controls the whole process. Changing of one bit only will result in quite divergent destination Data.

As an example we choose the start sequence:

"**Horse racing on the banks of San Sebastian**" (n = 42).

48 6F 72 73 65 20 72 61 63 69 6E 67 20 6F 6E 20 74 68 65 20 62
61 6E 6B 73 20 6F 66 20 53 61 6E 20 53 65 62 61 73 74 69 61 6E

The start sequence is a series of defined bytes **a(i)** with length **(n)**: message, string, key, character block, pixels, sounds etc. To work with any series of bytes as an object we have to assign it with a univalent value **H(k)**. Each byte **a(i)** has to be denoted with an index value and all **(n)** bytes have to be linked together in a qualified manner.

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

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

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

$$H(k) \;=\; 3901$$

In order to differentiate single bytes **a(i)** inside the message we have to consider additional significant features, otherwise we would'nt get definite results.

1 Expanding of Start Sequence to position weighted Interim Value Hk

With reference to **Renè Descartes** (1596 - 1650) we know that every object - which is scaleable in its dimensions - is exactly determined by its coordinates for **subject**, **location** and **time** (cartesian coordinate system).
We state:

object = **(m)** digital message of length **(n)**
subject = **a(i)** byte, element of the message
location = **p(i)** position of **a(i)** inside the message
time = **t(i)** time of **a(i)** inside the message

Time will be relevant only if there exists a functionally connection between a single byte **a(i)** and the clock frequency. May be in voice streaming but normally this connection is constant and we may set: **t = 1**.

In order to get an exact determination for each byte **a(i)** we associate **subject**, **location** and **time** by multiplying its dimension values and summarize all results to a destination value **H(k)**.

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

$$H(k) = 84208$$

Each byte **a(i)** is multiplyed with its position **p(i)** i.e. **position weighted** and thus consequently distinguished from each other. But collisions in consequence of exchanging bytes inside the message are not yet excluded. To achieve this we extend the value of a byte **a(i)** to a range superior of message length **(n)**. We attain this by extending **p(i)** with a constant **C(k)**. Derivation of "**hash constant C(k)**" you will find in internet at:

www.telecypher.net/Collfree.pdf

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

$$C(k) = 1681$$

Code is a number in the range from 1 to 99 in order to individualize the function.
We set: **code** = **1**

With embedding of „hash constant“ **C(k)** the interim value **H(k)** is calculated as follows:

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

$$H_k = 6641789$$

base 16:	**65587D**	base 62:	**Rrpd**
base 128:	**3Lm¬**	base 256:	**æä¬**

The result **H(k)** avoids collisions but is still to narrow to establish a secure unchallengable term. At the most it may serve as **MAC** for messages.

2 Expansion of Start Sequence onto Hash Function Series in Base 77 and Destination Value Hp

To obtain more provably security an **expansion function** is introduced which widens the destination factors onto utmost variables (about 160 up to 2400 digits). To achieve this the value of each byte **a(i)** combined with **p(i)** and interim value **H(k)** are expanded by multiplying into a higher range. The respective result **s(i)** is converted to **d(i)** a value in a superior number system (e.g. on **base 77**). The digits of **d(i)** are stored in serial manner in a **hash function series**. Simultaneous the procedure calculates the sum of all **s(i)** as an additional value **H(p)** in each round **r** ($r_1 \ldots r_j \ldots r_m$).

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

$$S_i \text{ ---> } d_i \quad \textit{(Basis 77)}$$

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

H(p) = 5559291769099

Chosen number system on **base 77** comprises the following digits:

0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz&#@àáâãäåæçèéêë
(determined by the author, not standardized)

In extracts single digits of **hash function series** are calculated as follows:

char		pi		Hk	(ai+1)*pi*Hk		Si	base 77
	(ai+1)		(ai+1)*pi			pi+code+r		
..	. . .	. .	. . . .	. . . . . .	. . . .	. .	. . . .	. . . . . .
t	117	17	1989	6641789	13210518321	19	13210518340	**4âzIAJ**
h	105	18	1890	6641789	12552981210	20	12552981230	**4n7PLK**
e	102	19	1938	6641789	12871787082	21	12871787103	**4wCnâL**
	33	20	660	6641789	4383580740	22	4383580762	**1IrãtM**
b	99	21	2079	6641789	13808279331	23	13808279354	**57zé0N**
a	98	22	2156	6641789	14319697084	24	14319697108	**5MRE0O**
n	111	23	2553	6641789	16956487317	25	16956487342	**6KRáoP**
k	108	24	2592	6641789	17215517088	26	17215517114	**6RuJKQ**
s	116	25	2900	6641789	19261188100	27	19261188127	**78æAKR**
..	. . .	. .	. . . .	. . . . . .	. . . .	. .	. . . .	. . . . .
					sum H(p)		559291769099	**2qmF6f#**

In our example the **hash function series** contains 248 digits in number system on base 77:

Dz293gO#Z4àEDD51ApWY61JRnk7bVht81ê7Uu91æ9ã7A2G3bsB2kLH2C2ërciD34zNfE14
4HtF3àJxEG46ièkH1MwdiI**4âzIAJ4n7PLK4wCnâL1IrãtM57zé0N5MRE0O6KRáoP6RuJKQ**
78æAKR288ZXS7WR9nT75äRuU2Q&vMV6E9xSW7YëzuX8t8nGY2pw9XZ70I0ga8wddEb8v
TGác8ä7GgdA&à0FeBF9ëWfAV7ã8g9áCC7hBX@LLi

The variables are digits (not characters) in number system on **base 77**. There is no way back to the original message (first one-way-function).

Simultanesly the program caclulates the following destination Data:

	decimal	base 77
hash constant C(k):	**1681**	**L@**
position weighted value (H_k):	**6641789**	**RgH0**
partial interim value (H_p):	**559291769099**	**2qmF6f#**
total hash value (H_p+H_k):	**559298410888**	**2qmTmw#**

Control parameters derived from destination Data show as follows:

Variante	**(H_k MOD 11) +1**	=	**1**	begin of contraction
Alpha	**((H_k + H_p) MOD 255) +1**	=	**134**	offset cipher alphabet
Beta	**(H_k MOD 169) +1**	=	**90**	offset block keys
Gamma	**((H_p + code) MOD 196) +1**	=	**37**	offset matrix key
Delta	**((H_k + H_p) MOD 155) +code**	=	**69**	dynamic bit series
Theta	**(H_k MOD 32) +1**	=	**30**	dynamic number series

3 Contraction of Hash Function Series by Modulo 256 to BASIC VARIATION

Next, to reduce the variables back to decimal data a **contraction function** (second one-way-function) is introduced. The digits of the **hash function series** are assumed to be digits in number system on **base 78** (expansion base +1). Each three digits of the function series are reconverted in serial manner by **MODULO 256** to decimal numbers 0 to 255 (without repetition). The results are stored in an array of 16x16 elemets: **BASIC VARIATION**. This function is not revisible and retrograde destination of preceding Data is not possible.

Reconversion to decimal numbers by MODULO 256 on **base 78** beginning at parameter **Variante** = **1:**

Dz293gO#Z4 . . . . . .

digit	decimal	MODULO		Element
base 78		256	- Theta	
Dz2	83852	140	30	**110**
z29	371289	89	30	**59**
293	12873	73	30	**43**
93g	55032	248	30	**218**
3gO	21552	48	30	**18**
gO#	257463	183	30	**153**
O#Z	150965	181	30	**151**
#Z4	386026	234	30	**204**
. . .	. . . . . .	. . .	. .	. . .

BASIC-VARIATION (256 elements)

Distribution of elements

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

The values of **BASIC VARIATION** are related directly to indexes of bytes with values **0** to **255** (analogous: ASCII-character set). During each round the procedure generates the „**CypherMatrix**“ with 16x16 elements from all elements of BASIC VARIATION either in three permutations or directly.

4 Permutation of BASIC VARIATION to CypherMatrix as final Hash Value

CypherMatrix (16x16) permuted from BASIC VARIATION (256 elements)

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

CypherMatrix in its structure (**GF 16^2**) constitutes a unique collision free term as "**Hash value**". Due to rules of probability a recurrence of an equal term will occur only in **256!** (faculty) = **8.578E+505** cases.

The control parameters to perform encryptions extract the following elements (bytes) out of the current CypherMatrix:

Alpha: ((K(k) + H(p) MOD 255)+1 = **134** defines the offset of extracting the cipher **alphabet** with 128 elements:

```
    1   EA    41 27       E6 F9 B9 7F 63 34    FE 47 CD    16
   17   D6 4B 6B 71 5E       44 BE    8A 87    E7 C0 A9    32
   33   FC 3F CC 85                                        48
   49   .. .. .. .. ..                      .. .. .. ..   128
  129                  C2 4E D1 31 7E 48 72 4F    5C      144
  145   8F 94 A2    68 C6 79 EF C5 F5 2A 29 F0    DA 81   160
  161   3C       33 53 46 9D 4D 62 2D BD EB 64 56 BF 26   176
  177   4A    A7       78 D3 8C F4 39    EE 69 2B F3 E2   192
  193   80 ED 5A FF 45 F7 36 76 73 5F B4 3A 55 C4 91 49   208
  209      B6 BA    77 A3 70 9C       FB    3B 9A EC F1   224
  225   F6       42 CE 4C AC 95       6C 66 D0 83         240
  241   D8 E4    75 61 37 25 D2 3D 2F    6E 8E 8D A1 CF   256
```

Certain characters (control characters **00** to **20, 22, 2C** and others) are excluded because in some situations they do still their original tasks (e.g. **1A** =ASCII-26) and will disturb the proper performance.

Beta: (H(k) MOD 169)+1 = **90** fixes the offset from where 63 bytes are taken to form the current **block key:**

```
   81                              6F 9F 59 AF E8 35 24    96
   97   99 51 B7 AD A6 C8 30 50 5D AB 60 65 32 E0 C3 67   112
  113   8B 14 90 E9 A8 84 3E 7A 98 A4 FD AE BC 6D 21 12   128
  129   5B 57 A5 A0 82 C2 4E D1 31 7E 48 72 4F 06 5C DB   144
  145   8F 94 A2 19 68 C6 79 EF                           160
```

Length of plaintext blocks - and by this length of block keys, as well - may be fixed alternatively with 35, 42, 49, 56, **63** or 70 bytes (multiple of 7) for each session. Block keys are necessary in encryption procedures with XOR-concatenations only.

Gamma: ((H(p) + code) MOD 196)+1 = **37** defines the offset for extracting the **matrix key:**

```
   33                  89 43 F8 DF 03 54 E5 96 2C B8 9E E1    48
   49   04 6A AA C7 38 86 D4 2E 07 7C 9B C1 D7 DD 92 08    64
   65   74 97 40 93 23 28 B3 01 52 0E B5 F2 CA E3          80
```

The matrix key will be led back (loop) to cycle's beginning in order to initialize the next run. The series of matrix keys controls the identical course of the complete procedure at sender and recipient, as well.

The sensitivity of the Basic Function is demonstrated in appendix "Changing the Start Sequence"

II. Coding Machine

Foregoing demonstration of the **basic function** comprises one cycle, only ($\mathbf{r_j}$) . To extend the procedure to a universally practicable cryptographic mechanism a new start sequence denoted as **„Matrix key“**(series of 42 bytes) is extracted out of the current CypherMatrix and lead back to the beginning ("**loop**"). This sequence initializes the next cycle ($\mathbf{r_{j+1}}$) and performs the demonstrated **basic function** once more. Consequently the number of repeated cycles will be unlimited.

Thus „**CypherMatrix**“ turns out to become a universally cryptographic device which may be used in most areas of cryptography as, so to say, „**coding machine**“. It serves for all control parameters which are necessary for cryptographic applications (e.g. matrix keys, cipher alphabets, block keys, byte stream, hash values, authentication marks, s-boxes etc.). The **matrix key** only controls the whole process in each round.

Basic function can be operated in extended form either with **10-bit** sequences in byte system on **base 10** as CypherMatrix GF(32^2) with 32x32 = 1024 elements or with **12-bit** sequences in byte system on **base 12** as Cypher Matrix GF(64^2) with 64x64 = 4096 elements.

Combined with the **basic function** especially the following applications have been developed completely in byte techniques.

Performing all kinds of encryptions with varied operations,
calculating of serial and final hash values (CypherMatrix Hash),
simple and extended signature (telePortal) and
generating of unlimited byte and number series.

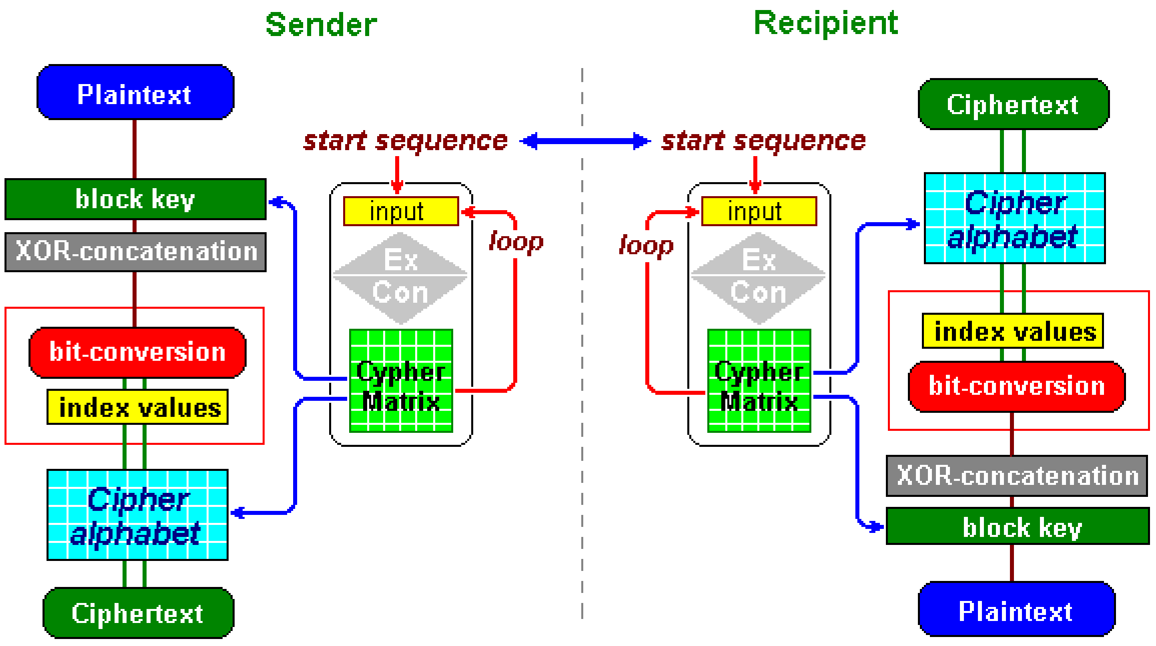
A. Encryption

In order to perform encryption the **basic function** is combined with qualified additional functions (**coding areas**). The respective **CypherMatrix** serves the control parameters necessary for processing encryptions. For each program the following parameters are needed:

length of >matrix keys<:	36 – 64	bytes
length of >block keys<:	35 – 91	bytes
numbersystem for expansion function:	35 – 96	basic digits
individual user code:	1 - 99	numbers

To perform encryptions the following scheme shows the combined working of **basic function** and **coding area:**

Encryption / Decryption scheme



As symmetric procedure both **sender** and **recipient** insert an identical **start sequence** which constructs and controls the whole process. This passphrase has to be easy to remember and possibly should be catchy and even of funny words which one can easy keep in mind and which may not be written down anywhere. Here are some examples:

Sven Hedin is sailing around the Northpole	[42 bytes]
Kangaroos jumping up the Hills of Amarillo	[42 bytes]
Blue flamingos flying to Northern Sutherland	[44 bytes]

Start sequences should be about **42 bytes** long. Hence, because of their length lexical attacks and iterative searching should be impossible. An attacker even isn't able to analyse parts of the start sequence neither apart nor successive because the passphrase could be found in total only, if at all.

To make encryption more secure a **„Bit Conversion“** is introduced. It is a step which regroupes the results of XOR concatenation from 8-bit segments into 7-bit segments (values from 0 to 127). The rows of digits „0“ and „1“ remain unchanged. A new grouping (8-bit → 7-bit: changing the bytesystem, bit conversion) is arranged, only.

8-bit XOR-sequences rearranged into 7-bit sequences as "**index data**" for the **cipher alphabet** of 128 characters.

8-bit: **11101111100001111001101010101111101010011101111101000100** ....
7-bit: **11101111100001111001101010101111101010011101111101000100** ....

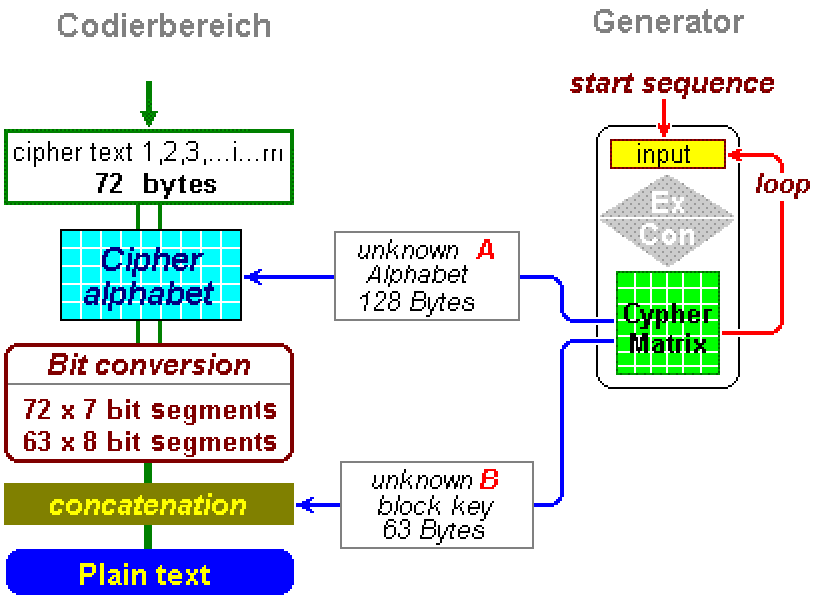
The separated 7-bits are **indexes** (pointers) only to positions in that array **cipher alphabet** which has been extracted from the respective CypherMatrix**.**

The process is similar to **coding base 64** where 8-bit segments are converted to 6-bit segments which each get a specific cipher character from a 64 elements alphabet (changing from byte system on base 8 to byte system on base 6).

The encryption process is performed as a **dynamic one-time-pad** because by each cycle a new **block key** of **63 bytes** which has the same length as the respective **plain text block** is extracted out of the current CypherMatrix and no block key will be identically repeated in further cycles.

Security of the encryption process may be judged by the following facts: Principally an attacker knows the **Ciphertext** and the respective **length** of the encrypted message, only. Further he may know the "CypherMatrix" method with its three functions, but he does'nt know the actual control parameters: **Alpha**, **Beta** and **Gamma**. Because an attacker even does not know the **start sequence** he will be unable to break the cipher.

1. **Plaintext --> block key --> XOR-concatenation,**
2. **8-bit XOR-concatenation --> 7-bit index values (0...127) and**
3. **7-bit index values --> ciphertext array (0...127) --> ciphertext.**



The parameters **block key** (**B**) and **ciphertext array** (**A**) are two variables independent from each other.

Cipher text = **f [** f1 (plain text,**k1**), f2 (b1,b2), f3 (b2,**k2**) **]**
Plain text = **f [** f3 (cipher text,**k2**), f2 (b2,b1), f1 (b1,**k1**) **]**

fx = function
k1 = block key
b1 = 8-bit sequence
b2 = 7-bit index-value
k2 = cipher text array (128)

Both, encryption and decryption form equations with two unknown variables: **k1** and **k2**. This results in a definite solution only if one of the unknown can be derived from the other unknown variable or if there are two equations with the same unknown variables. But there is no connection between the respective block key= **k1** and the array (cipher alphabet 128)= **k2** generated in the same cycle. Both are extracted out of the current CypherMatrix but they have no functional connection (**k1** → (**Hk MOD 169**)+1) and (**k2** → (**Hk+Hp MOD 255**) +1). Both their common roots are the initial **start sequence** only. But to that position is no way back (two one way functions prevent this).

B. Hash Function

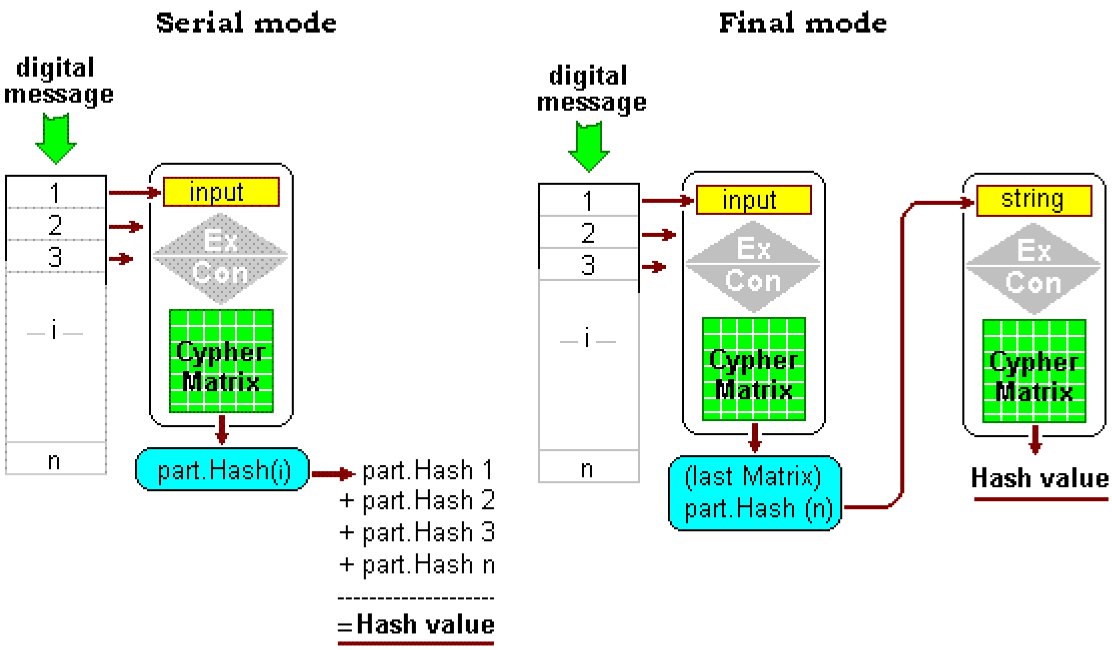
Digital strings (files) are devided into plaintext blocks (e.g. 64 bytes), which in each cycle will be sequentially position weighted, multiplied with the hash constant **Ck,** expanded to hash function series, again contracted to BASIC VARIATION and finaly subjected to a threefold permutation. The resulting last CypherMatrix with 256 elements represents the **Hash Value (H)**.
An identical hash value will occur first in **256!** (faculty) = **8E+506** cases.
Compared with current hash functions the procedure has some advantages:

1. Including of a **compression** is not necessary,
2. only bytes and simple mathematics are implemented,
3. the results of the function (hash values) can be **calculated** (e.g.. addition, subtraction, multiplication, division und modulo calculation) and
4. the results are remarkable **shorter** than 128 bit resp.160 bit (SHA, MD5, RIPE-MD).

On principle two techniques are possible:

a) Serial calculation of partial hash values from the CypherMatrix in each round and accumulating to a hash value ad end (**serial mode**) and
b) final calculation of the hash value from the last CypherMatrix beyond all foregoing cycles passed (**final mode**)

Connections are shown by the following schemes:



As an example of a text file >CANNERY.TXT< (401 bytes) the function calculates in „**final mode / last cycle**“ the following hash values:

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

decimal	base 62	cycle	CM(k)
9179333265149120	g2ZJCcxeq	1	2L3OW0
8875455018876196	eeHMUbp8m	2	2L2nhi
9025823595423872	fKygWexbU	3	2L35v6
8632629465403536	dXKHWvnXs	4	2L2KHk
9148458703148928	ftnkEhEa8	5	2L3Km8
9292204181825284	gYcSrSf80	6	2L3cBe
9214602417074564	gCaF3qA6u	7	2L3Sms

----

decimal: 63368506646901500
base 16: E1215230A2B2FC
base 62: 4gEA5upbm0 **H(s)**

Matrix value last cycle:
decimal: 8823877576185252
base 16: 1F5945187855A4
base 62: ePdJPAY44 **H(l)**

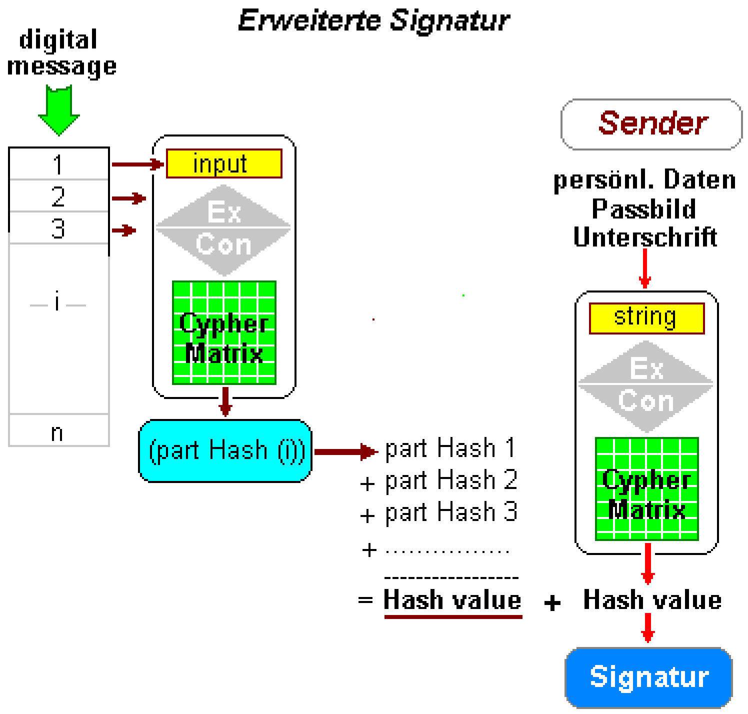
Final Hash value for: CANNERY.TXT
decimal: 72192384223086752
base 16: 1007A97491B08A0
base 62: **5KdnPK09q4** **H**

=====================================

Development and working features of CypherMatrix Hash Function are detailed demonstrated in the following pdf-file: http://www.telecypher.net/DynahashE.pdf

C. Extended digital Signatures

If the hash function is applied to both the **message** to be **signed** and the **identification data (ID)** of the signatory (including personal photo and signature) an **„extended digital signature"** can be created.



Next, an individual example is demonstrated:
(Note: name and data are free devised)

Name, location:	*Geoffrey Durbridge, Minnesota*
birthday, location:	*14.07.1954 / Stratford-upon-Avon*
pers. passphrase:	*Blue flamingos flying to Northern Sutherland*
passport number:	*890-abc-1234567*

		base 62	decimal
hash value pers. Data (base 62):		krPSUVaf5	10231736539462315
passport photo (passfoto.jpg):	+	12eyYLjGJD	14118070825423503
digital Sign (mysign.jpg):	+	pTlltgq91	11240185700633983
ID-Data (base 62):	=	2d09mjvh7J	35589993065519801

If the ID-Data of the user will be added to the hash value of any file we get a simple digital signed document:

	base 62	base 16	decimal
sender's ID-Data	2d09mjvh7J	7E70E8FE3A9AB9	35589993065519801
+ file Cannery.txt	+ liR6AtL4G	+ 25038C4D86B744	+10418475269273412

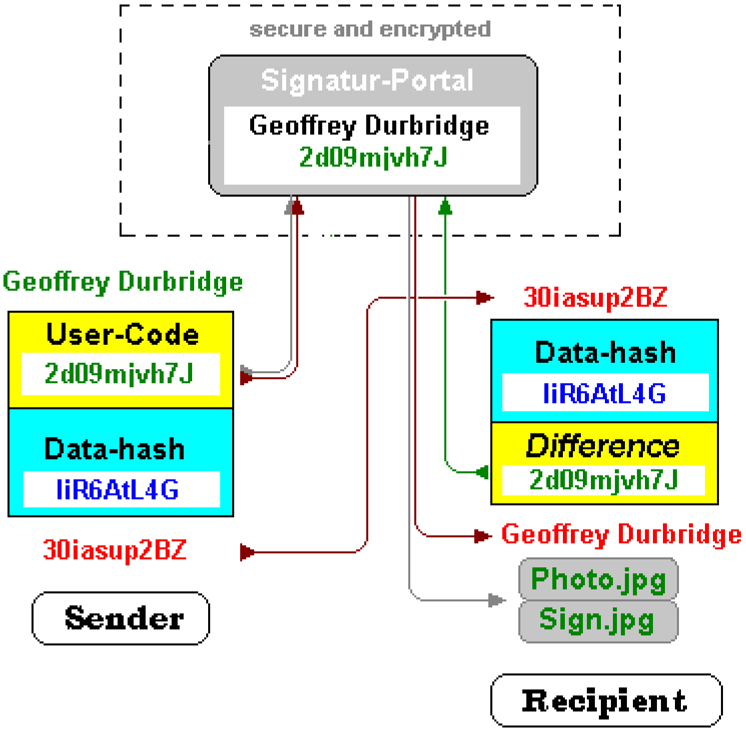
	----------------	--------------------------	----------------------------
digital signature	30iasup2BZ	A374754BC152FD	46008468334793213

The **digital signature** is a determined information about content and sender in comprehended form. If the digital signature is transferd together with the respective file the recipient

a) is able to calculate the **hash value** of the file and
b) may verify "senders **ID-Data**" from the difference between hash value and the received "digital signature".

If sender's ID-Data is stored in an internet database ("Signatur-Portal") – of course secure and encrypted - the recipient may test the integrity of the message and the identity of the sender by a checkback to the internet database.

Data stored in the internet database are secured that way that the owner of the signed message only can call and decrypt the Data. Nobody else has access on the stored Data.



Besides the personal Data of the signatory his digitally photo and his digitally sign are included into calculation of the hash value. A recipient of the signature will be able to verify the integrity of the message and the identity of the sender. He even may download the pass photo and the personal sign of the sender and verify it by himself.

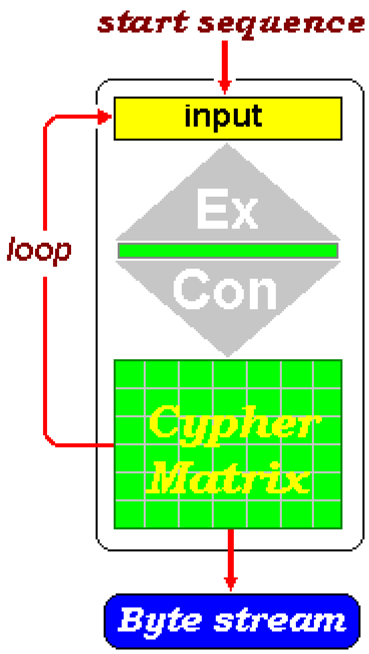
More details you will find in internet in the file: Eine einfache erweiterte Signatur

D. Unlimited Byte Series

The **basic function** produces unlimited **bit-** and **byte series** (byte stream). In each round all 256 elements of the respective CypherMatrix are stored correspondingly their distribution in succesive lines (byte series). Length of these series depends on chosen number of rounds. As "**random number series**" they are suitable for many-sided cryptographical tasks (e.g. as key files for "one-time-pad" and "multi-time-pad" encryptions).

If otherwise **contraction** is performed with **MODULO 8** (alternatively with **16, 32, 64** or others) instead of MODULO 256 (contraction to BASIC VARIATION) any number series may be created, for instance for encryption operations.

Byte Generator



The well known tests „**ENT.exe**“ and „**FIPS PUB 140-1**“ of U.S.NIST recognize that the procedure meets most of all requirements of **random generating** bytes. By changing of only **one bit** in the **start sequence** there will be no corresponding byte left at same positions in further generated series.

More details about bytes used in cryptographic solutions you will find in internet:

www.telecypher.net/INDENG.HTM

From there you may download several DEMO programs and test all by yourself.

Munich, in January 2010
Ernst Erich Schnoor