MEMO

Paradigma change in Cryptography

(Ernst Erich Schnoor)

Cryptography means writing and reading of secret messages by way of characteristic methods (encrypting). Since computers came up and took possession of crytrography **bits** and **bytes** are confusing users. Hence, it seems to be such simple: 8 bits are 1 byte. That assumes uniformity . But, it is still possible to create various techniques in cryptography. Analysing lengths between **plaintext** and **ciphertext** shows that in principle both have identical length (without header, control checks, padding, time stamps etc.). Both texts should be of equal length, because ciphertext has to be stored at the same space where plaintext had been stored before [#1].

If both texts are of equal length then their number of characters (**bytes)** equal, as well [#2,#3]. Each plaintext character corresponds with one specific cipher character, in fact only **one.** Encryption from one plaintext character to one cipher character works by a **functional** changing of 8 bits inside of one byte (**bit series**), even if the procedure passes longer bit sequences than 8 bits (e.g. 32, 64 bits). Because characters (**bytes**) in plaintext and ciphertext as well comprise 8 bits then there must be a uniform system (naturally 8 bit) and an identical system alphabet of 256 signs (**paradigma** of cryptography). But restriction to 8 bits seems to be one-sided and too narrow. Obviously real important phenomenons are lacking: **systematics** of bit series und **system alphabets** for realizing visibility of information contents.

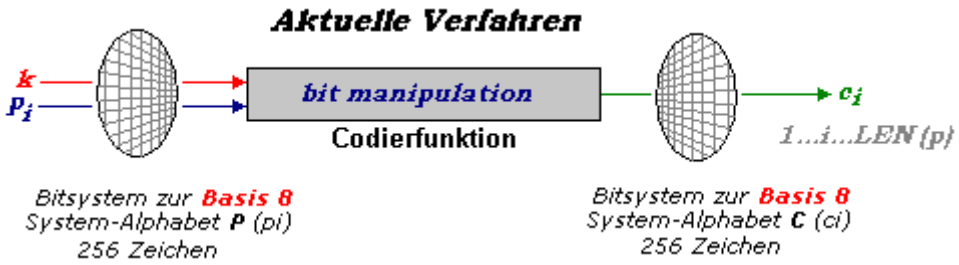
1 Systematizing of bit series

In data technical respects a computer only processes sequenzes of „0“ and „1“: **bit series.** In their quantity they are unlimited (1 to ∞). In order to work with bit series , realizing interdependences, searching for connections and creating programs – in particular enciphering – bit series have to be devided into definite segments (**units**), that means: they have to be systematized. It is a similar phenomenon compared with the quantity of all numbers. Comparable to numerical theory bit series can be systemized in a significance order system. Bits comply with digits, units (resp. bytes) conform with numbers and the system alphabet represents the ordered number quantity. Hence, 8 bit sequences may be named as „**bitsystem on base 8**“. In particular:

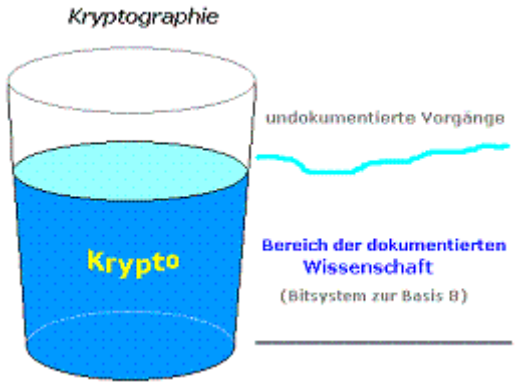
			system alphabet
bit sequences:	1-bit = bitsystem on base 1	= 2^1 signs =	2 units
	2-bit = bitsystem on base 2	= 2^2 signs =	4 units
	3-bit = bitsystem on base 3	= 2^3 signs =	8 units
	4-bit = bitsystem on base 4	= 2^4 signs =	16 units
	5-bit = bitsystem on base 5	= 2^5 signs =	32 units
	6-bit = bitsystem on base 6	= 2^6 signs =	64 units
	7-bit = bitsystem on base 7	= 2^7 signs =	128 units
	8-bit = bitsystem on base 8	**= 2^8 bytes =**	**256 bytes**
	9-bit = bitsystem on base 9	= 2^9 signs =	512 units
	10-bit = bitsystem on base 10	= 2^10 signs =	1024 units
	11-bit = bitsystem on base 11	= 2^11 signs =	2048 units
	12-bit = bitsystem on base 12	= 2^12 signs =	4096 units
	16-bit = bitsystem on base 16	= 2^16 signs =	65536 units
	32-bit = bitsystem on base 32	= 2^32 signs =	4294967296 units

Up to now following bit sequences have been defined as units: bitsystem on base 1 comprising the signs: „0“ and „1“ or historically: „L“ and „H“. Bitsystem on base 4 knows the unit „nibble“ and 8-bit sequences the unit: „**byte**“. In bitsystem on base 16 are the units: „word“ and „dword“.
All other bit series are marked by their number of bits (e.g. 32 bits, 64 bits) [#4].

By introduction digitally techniques the scientists appointed to further development superficially relaid upon bitsystem on base 8 and obviously rated an appropriate systemizing as subordinated..
Almost all operations are performed in bit system on **base 8.** All inputs are processed in bitsystem on base 8 and system-alphabet on base 8 with 256 signs (00 to FF). The encrypted results (cipher text) are shown in characters of bitsystem on base 8 and system-alphabet with 256 signs, as well. In view of the fact that between input and output there are manifold manipulated longer bit series it is of no significance. Insofar – independent of manipulated bits inside function from input to output – all coding operations perform in a uniform **order system**, in bit system on **base 8**.



With the described connections and the demand plaintext and ciphertext must have the same length the **action range** of cryptography will be limited systematically to a more restricted extend:
bitsystem on **base 8** with a system alphabet (cipher alphabet) of **256 bytes**. Obviously as a consequence of this limitation more real existing phenomenons will not be considered. The cryptographic scientific discipline shows some undocumented areas at this point, so to say a **„crypto incognito“**.



By systemizing bit sequences and assigning system alphabets a new part of encryption techniques becomes visible: **bit conversion** and **system alphabets** (new **paradigma** of encrypting).

By new consideration with system orientated bit segments and system-alphabets on base **1** up to base 12 (and higher) a number of new possibilities will be available for practice and research, as well. In this area is introduced the **CypherMatrix** procedure (name by the author) [#13], which will be explained in the following sections. Top content of the procedure is converting bit systems and generating appropriate system alphabets.

2 Bit Conversion

Bit conversion is changing bit sequences from one bit system to another bit system. Number of bits and their order remain unchanged. No bit is added and no bit is removed. The structure will be changed, only. Decimal values of the new units serve for indexation to the assigned system alphabet. A bit conversion may transact for all bit systems from base 1 to base 12 (and higher).

Fundamentally connections are shown in following scheme:

System base	System-alphabet	Bit series in Units indexing	Lengths ratio
bitsystem on base 1	2	0 1 0 0 1 1 1 0 0 1 1 0 1 1 1 1 0 1 1 1 0 1 0 0 0 1 1 0 0 0 0 0.	1:8
bitsystem on base 2	4	01 00 11 10 01 10 11 11 01 11 01 00 01 10 00 01 01 10 00. 1 0 3 2 1 2 3 3 1 3 1 0 1 2 0 1 1 2 0	1:4
bitsystem on base 3	8	010 011 100 110 111 101 110 100 011 000 010 110 001 ... 2 3 4 6 7 5 6 4 3 0 2 6 1	1:2,66
bitsystem on base 4	16	0100 1110 0110 1111 0111 0100 0110 0001 0110 0010 ... 4 14 6 15 7 4 6 1 6 2	1:2
bitsystem on base 5	32	01001 11001 10111 10111 01000 11000 01011 00010 01.. 9 25 23 23 8 24 11 2	1:1,6
bitsystem on base 6	64	010011 100110 111101 110100 011000 010110 001001 ... 19 38 61 52 24 22 9	1:1,33
bitsystem on bases 7	128	0100111 0011011 1101110 1000110 0001011 0001001 ... 39 27 110 70 11 9	1:1,143
bitsystem on base 8 *operations*	256	01001110 01101111 01110100 01100001 01100010 011.. 78 111 116 97 98 4E 6F 74 61 62 N o t a b ... *streamcipher, blockcipher with congruence of length, Feistel-nets* ECB. CBC. CFB. OFB. DES. IDEA. AES, RSA, *differential and lineare cryptanalysis, and other programs*	1:1
bitsystem on base 9	512	010011100 110111101 110100011 000010110 001001100 156 445 419 22 76	1:1,79
bitsystem on base 10	1024	0100111001 1011110111 0100011000 0101100010 01100 313 759 280 354	1:1,6
bitsystem on base 11	2048	01001110011 01111011101 00011000010 11000100110 ... 627 989 194 1574	1:1,46
bitsystem on base 12	4096	010011100110 111101110100 011000010110 001001100 1 1254 3956 1558 613	1:1,33
bitsystem on base xx	*div*	x x x x x	1: xxx

Up to now changing of bit sequences are performed in procedure **„Coding Base 64“,** only [#5]. 8-bit sequences are converted to 6-bit segments which each get a specific cipher character from a 64 elements alphabet (changing from bitsystem on base 8 to bitsystem on base 6). The system alphabet is stated fix.

According to the above prinziples the **„CypherMatrix“** procedure [#13] is developed. A couple of programs (67) by CypherMatrix procedure are shown in paragraph 4.1 of this article (program overview). They may download from the author by e-mail with or without source code.

2.1 System alphabet

The system alphabet is the most important component of computer techniques. It establishes foundation for visualization of bit series content. Without definition of an appropriate **alphabet** the computer will not be able to work, at all.

The ASCII-character set in its respective charateristics is the fundamental system alphabet. For more voluminous character sets – especially foreign languages – **unicode** is used. Each computer output requires a concerning system alphabet (array). So, for example, for images (pixel, graphical programs), sounds (digital/analogous converter, audio files), digital measuring instruments, bar coding and all further outputs of structured phenomenons in scale analysis and bit presentation.

2.2 Conversion Base 1 to Base 8

Handling of bit series in segments of (historical) 7, further (extended) 8 bits, fitted out with an index of decimal numbers (0 to 127 resp. 0 to 255) and combined with a specific system alphabet (several ASCII-character sets with 256 signs) already represents an essential encryption. But, it is not recognized as such an operation.
Conversion from bit system on base 1 to base 8 operates in particular as follows:

system alphabet:		decimal	hexadecimal
alphabet$(98)	=	b	62
alphabet$(101)	=	e	65
alphabet$(102)	=	f	66
alphabet$(103)	=	g	67
alphabet$(105)	=	i	69
alphabet$(108)	=	l	6C
alphabet$(110)	=	n	6E
alphabet$(111)	=	o	6F
alphabet$(116)	=	t	74

Converting following bit series:

bit series base 1:
01100010011010010111010001100110011011111011011000110011101100101011011110
bit series base 8:
01100010 01101001 01110100 01100110 01101111 01101100 01100111 01100101 01101110
index: 98 105 116 102 111 108 103 101 110
system alphabet (ASCII-character set):
b i t f o l g e n
index (hexadecimal) = system alphabet (hexadecimal):
62 69 74 66 6F 6C 67 65 6E
In plaintext the bit series reads: „**bitfolgen**“
Hexadecimal output reads: „**626974666F6C67656E**“

The original bit series on base 1 remains unchanged. This example makes clear that only the system alphabet has to be changed whereas the unstructured origin bit series (number and order) remain unchanged. All conversions operate by this manner.

2.3 Conversion Base 8 to Base 7

Following example demonstrates a conversion from base 8 to base 7:

bit series base 8:
01100010 01101001 01110100 01100110 01101111 01101100 01100111 01100101 01101110
index: 98 105 116 102 111 108 103 101 110
system alphabet:
b i t f o l g e n
bit series base 7:
0110001 0011010 0101110 1000110 0110011 0111101 1011000 1100111 0110010 1011011 10
index:
49 26 46 70 51 61 88 103 50 91
(+1) 50 27 47 71 52 62 89 104 51 92
cipher alphabet:
> f Q < µ ¸ 8 ‡ Å
In bitsystem on base 7 ciphertext is created as follows: **>fQ<µ¸8‡Å**
Such processing seems to be the best way for encrypting: 9 plaintext characters lead to 10 ciphertext characters.

System alphabet on base 7 with 128 units created by generator of **CypherMatrix** inserted with start sequence: „Checkpoint Charly at Medison Square Garden“ reads as follows.

Cipher alphabet (base 7)

1	õ Ò a C y Ð k S c ª N › © ¼ . ?	*16*
17	U ÷ ˆ Z è Ë g b f s ô (+ ð	*32*
33	″ â ¶ ° À – ¿ ` ¯ d Ñ ê Ó 9 Q Ù	*48*
49	) > ‡ µ K P ‟ ' ½ h ò ù í ¸ Ê W	*64*
65	È R € œ e t < á 0 ‹ X Ç ¨ o H ä	*80*
81	Ã – J ; « ö [ñ 8 Ì $ Å E ü , †	*96*
97	¾ | 6 Ö q ¹ Ž j ´ T • Y { l	*112*
113	û 5] Í Î V „ : Š É § é ¡ ï Â	*128*

2.4 Conversion Base 8 to Base 12

Alternative ciphering is demonstrated by a conversion from bitsystem on base 8 to bitsystem on base 12. Number of bits and their order remain unchanged. No bit is removed and no bit is added. Only spaces between the units vary. 8-bit sequences change to 12-bit sequences. Decimal values of the new units are index values to the signs in system alphabet on base 12. The system alphabet will be created by CypherMatrix procedure. Here 4096 signs are requested.

Because lack of single signs the digits of number system on base 64 (4096 elements) are used as double-characters.

. . . . . .
alphabet(1390) = jU
alphabet(1574) = gc
alphabet(1638) = fc
alphabet(1654) = fM
alphabet(2420) = TO
alphabet(3948) = 5W
. . . . . .

Bit conversion from base 8 to base 12 operates as follows:

characters base 8:

b	i	t	f	o	l	g	e	n
index: 98	105	116	102	111	108	103	101	110

bit series base 8:

01100010 01101001 01110100 01100110 01101111 01101100 01100111 01100101 01101110

bit series base 12:

011000100110 100101110100 011001100110 111101101100 011001110110 010101101110

index:	1574	2420	1638	3948	1654	1390
system alphabet:	gc	TO	fc	5W	fM	jU

In bitsystem on base 12 the encrypted „bit series“ read as follows: „**gcTOfc5WfMjU**“

9 plaintext characters convert to 6 double-signs, respective 12 ciphertext characters.

2.5 Consequences of bit conversion

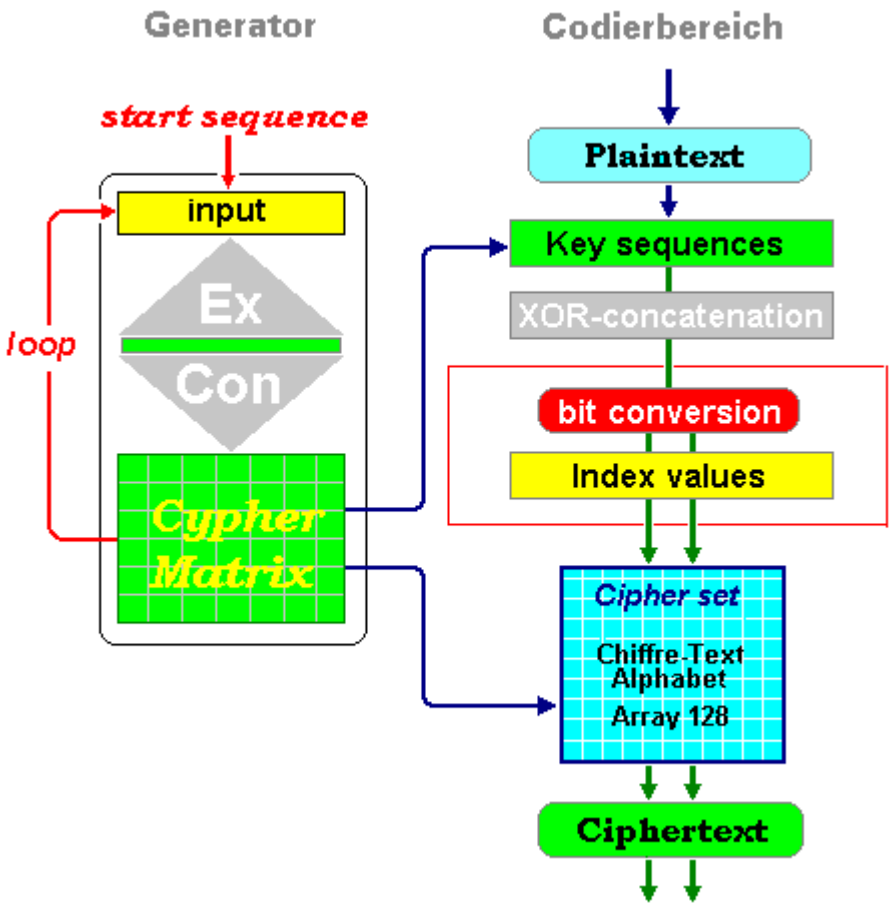
As a fundamental result of bit conversion turns out: Ciphertext becomes longer in lengths ratio: plaintext length in original bitsystem to ciphertext length in aimed bitsystem. Demanding plaintext and ciphertext should have the same length is principially not fulfilled.

In case of bit conversion there is no uniform **order system** between plaintext and ciphertext. For instance by changing from bitsystem on base 8 to bitsystem on base 7 each plaintext character leads to a ciphertext character which is **1,143** times longer than the origin.
Cryptanalysis by certain steps (word combinations, repetition patterns, frequency structures, two-digit groups, differential and linear analysis [#12]) presuppose plaintext and ciphertext at a ratio of **1:1**. Due to missing congruence of length both texts can not be compared with another. Attacks on **CypherMatrix** procedures based on this features will have no success.

For following encryption steps like Feistel networks [#6], shift steps, s-boxes, ECB, CBC, CFB and OFB, confusion and diffusion and further operations the conditions are missing. There are no uniform order system and no congruence of length. The mentioned encrypting steps cannot be used any longer.

3 Encryptions by „CypherMatrix“

Contrary to today's traditionally used algorithms **CypherMatrix** goes its own way. A generator creates the parameter neccessary for processing encryption. Coding itself is performed in a separate „**coding area**“.



Both sectors are combined together, but can be used separate, as well. The essential task of the generator is to serve with all required parameters necessary for encryption. Actual processing takes place in the coding area.

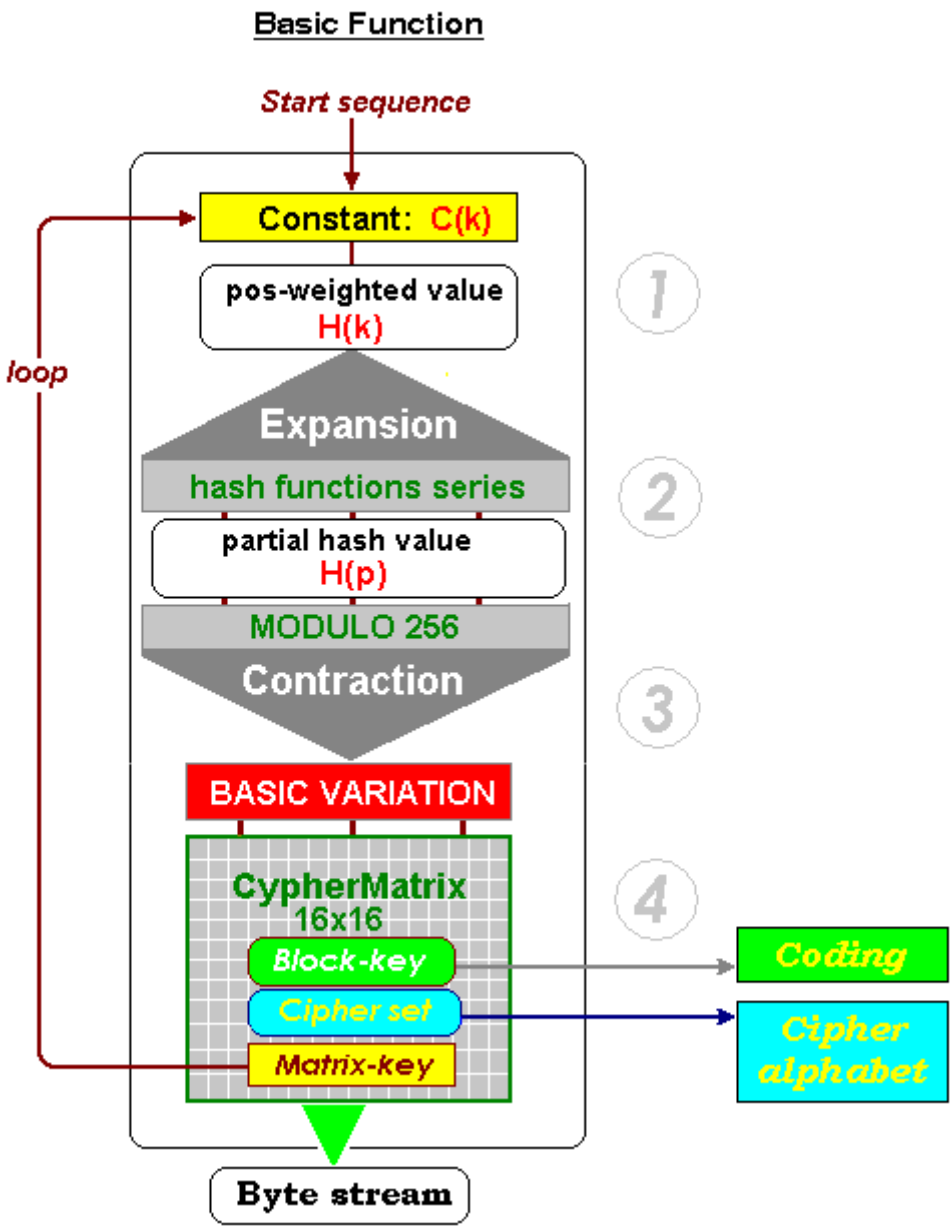
3.1 The Generator

The procedure ist arranged as block cipher. It is symmetric, because sender and recipient have to insert the same start sequence in order to initialize the generator. The procedure is dynamic, because the generator creates new control parameters for each plaintext block (e.g. 63 bytes) in each round. Any start sequence (passphrase) with at least 36 and optimal 42 characters drives and controls the whole process. Some examples:

Horse racing on the banks of San Sebastian	[42 bytes]
Koala bears are diving at Murray's Mouth	[40 bytes]
7 kangaroos jumping along the Times Square	[42 bytes]
Blue flamingos flying to Northern Sutherland	[44 bytes]

The start sequence should be easy to remember and possibly somewhat catchy and of funny words. The chosen phrase cannot be guessed, can easy be kept in mind and may not be written down, anywhere. 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.

Each start sequence at sender and recipient, as well, generates identical control parameters and course of the procedure.



For each round the generator delivers all control parameters necessary for cipher processing.

1. Cipher alphabet (system alphabet) for the actual round,
2. block key for XOR-concatenation and
3. matrix key as start sequence for the next round.

At the end of each round a **CypherMatrix** with 16x16 elements is created, which serves for all control parameters to achieve the encryption. The matrix key (42 signs) is to be led back to the beginning (**loop**) in order to initialize the next round, till an end mark is designated. Due to probability laws a repetition of identical Data will occur first in **256!** (faculty) = **8E+506** cases .

3.2 Inserting Start Sequence

The following start sequence (input) is chosen as an example:

Checkpoint Charly at Madison Square Garden [42 bytes]

```
43 68 65 63 6B 70 6F 69 6E 74 20 43 68 61 72 6C 79 20 61 74 20
4D 61 64 69 73 6F 6E 20 53 71 75 61 72 65 20 47 61 72 64 65 6E
```

The goal is to find an evident determination base for analysing the start sequence.
Input **m** is a series of definite bytes **a(i)** with lengts **n.** In order to analyse the series as state of facts the single characters have to be systematized (scaled). For this each byte **a(i)** gets an index and all bytes will be appropriate associated with each other (addition).

$$\mathbf{m} = a_1 + a_2 + a_3 + \ldots a_i + \ldots a_n$$

(single value of „a(i)“ has to increase by (+1) because otherwise ASCII-zero (0) would not be considered)

$$\mathbf{m} = \sum_{i=1}^{n} (a_i + 1)$$

$$\mathbf{m} = 3991$$

In order to differentiate single bytes **a(i)** inside the series additional criterions have to be added, because otherwise no definite results will resume.

3.3 Extending to Position Weighting

With reference to **René Descartes** (1596-1650) we know that every object (fact) – which is scalable in its dimensions – can be exactly determined by coordinates for **subject, location** and **time** (cartesian coordinate system). We set:

object: (**m**) digital information of length (**n**)
subject: (a_i) elements of information, signs, bytes
location: (p_i) position of byte **a**(**i**) inside the information
time: (t_i) point of time of byte **a**(**i**) inside the information

In order to distinguish single characters each byte **a**(**i**) is multiplied with its location **p**(**i**). Time will be relevant only if there exists a functionally connection between a single byte and clock frequency. Normaly this connction is constant and we may set: **t = 1**. In order to get an exact determination for series **m** we associate **subject**, **location** and **time** by multiplying its dimension values and summarize all results to a destination value **H**(**k**):

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

$$\mathbf{H(k)} = 85589$$

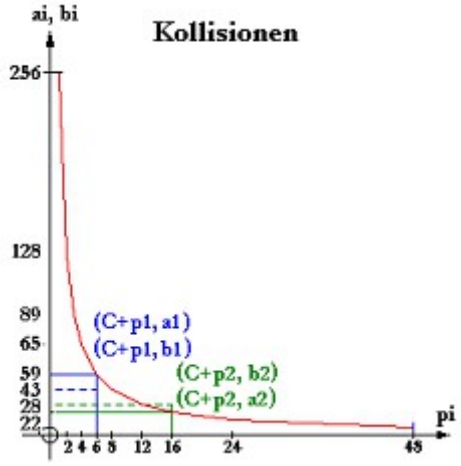
With position weighting the single bytes differ but collisions in consequence of exchanging bytes inside the series are not excluded, yet.

3.4 Excluding Collisions

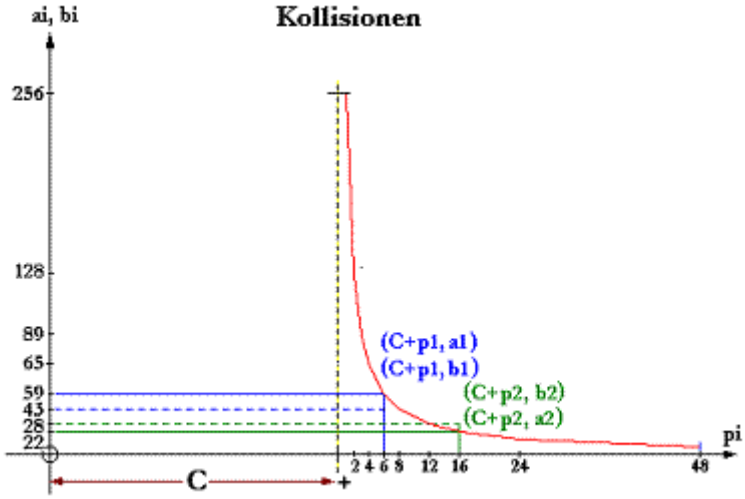
A collision occurs at following conditions:

$$\text{collision:} \quad \mathbf{H(k)\ a_i} = \mathbf{H(k)\ b_i}$$
$$(\mathbf{a_1}+1) * \mathbf{p_1} + (\mathbf{a_2}+1) * \mathbf{p_2} = (\mathbf{b_1}+1) * \mathbf{p_1} + (\mathbf{b_2}+1) * \mathbf{p_2}$$

At position $\mathbf{p_1}$ the sign $\mathbf{a_1}$ is exchanged with the sign $\mathbf{b_1}$ and at position $\mathbf{p_2}$ the sign $\mathbf{a_2}$ with $\mathbf{b_2}$.
The following graph illustrates the connections between single signs $\mathbf{a_i}$ and $\mathbf{p_i}$ in multiplication formula $(\mathbf{a_i}+1) * \mathbf{p_i}$.



To avoid collisions the position weighting is shifted by a distance **C** to an area above length **n,** that means: $\mathbf{p_i}$ will be extended by a constant distance **C.**



$$(a_1+1) * (C+p_1) + (a_2+1) * (C+p_2) = (b_1+1) * (C+p_1) + (b_2+1) * (C+p_2)$$

After transforming we get the changing quotient **Q**:

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

For the „changing quotient“ – here denoted with **Q –** three cases are relevant:

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

If $Q = 1$ then $(C + p_1)$ and $(C + p_2)$ must be equal, as well. Because exchanging of characters occured at the same position **p** here is a collision excluded. If $Q > 1$ or $Q < 1$ then $(b_2 - a_2)$ and $(a_1 - b_1)$ are different, as well. Because a_1, a_2, b_1 and b_2 being integer values their differences will be integer values as well.

Positions p_i in the start sequence with **N** bytes (length n) cover a range from **1** (*minimum*) to **N** (*maximum*). Such the changing quotient of above formula comprises the following range:

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

Further development is demonstrated in article:

"Determinants leading to collisionfree"

The result leads to following formula: $C = N * (N - 2)$

Factor **C** depends on length **N** of the start sequence only. **C** includes the properties being equal for all start sequences with same lengths and deviding values of position weighting in collisionfree and collision burdened segments. Because of this the factor **C** gets the name: dividing constant **C**(**k**).

In order to individualize the function an additional code is introduced – a chosen number between 1 and 99. We set code = 1:

$$C(k) = n * (n - 2) + code$$
$$C(k) = 1681$$

By including the dividing constant **C**(**k**) the destination value H_k results as follows:

$$H_k = \sum_{i=1}^{n} (a_i + 1) * (p_i + C_k + round)$$
$$H_k = 6798451$$

The result **H**(**k**) avoids collisions but is still too narrow to establish anvulnerable destination values. Probably it could serve as **MAC** for messages.

3.5 Extending to Hash Function Series

To extend the destination base an **expansion function** is introduced which widens the sequence to an extensive series in a superior number system. The number system of expansion may be chosen betwenn 64 and 96. Here the number system on **base 77** is fixed. The function calculates for each value of the inserted sequence its decimal value s_i which is changed to d_i **-** digits in number system on base 77. Simultaneous the function calculates the sum of all single results s_i as an additional destination value **H(p)** for creating several control parameters and accumulates the results d_i serial as hash function series (HF).

$$s_i = (a_i + 1) * p_i * H_k + p_i + code + round$$
$$s_i \rightarrow d_i \quad (base\ 77)$$

$$HF = d_1 + d_2 + d_3 + \ldots + d_i + \ldots d_m$$
(m = number of digits in system on base 77)

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

$$H_p = 581872623626$$

The chosen number system on base 77 comprises the following digits:

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

By generating hash function series the sequence "**Charly**" at position 12 of the inserted passphrase results to the following calculations:

char	pi		Hk	(ai+1)*pi*Hk		Si	Basis 77
	(ai+1)	(ai+1)*pi			pi+code+r		
C	68	12 816	6798451	5547536016	14	5547536030	23&YPa
h	105	13 1365	6798451	9279885615	15	9279885630	3Wêáx F
a	98	14 1372	6798451	9327474772	16	9327474788	3YQ8LG
r	115	15 1725	6798451	11727327975	17	11727327992	4Pkzeç
l	109	16 1744	6798451	11856498544	18	11856498562	4TLuw&
y	122	17 2074	6798451	14099987374	19	14099987393	5G7æGU
					Summe:	581872623626	2yégr4#

The hash function series comprises 248 digits in number system on base 77:

DBlxäelGF4xDyëæ10Rjdo1RXUnp1s9Wbf1élåu929ëéæs2dFjB02çL2HuåFkrk23&YPa3W
êáxF3YQ8LG4Pkzeç4TLuw&5G7æGU1bâfVr4q7æuL5âg4ZX1v1zBN4Náxuã5oåU7P66yçfb
6ocW2n7iLroz7j##&T7&5sãU2V6B8K6PReQW8âZbAá9bKGçC89Y30Z9#DqFw8éWiOb2êw5
#R6rFä3z9RFáZeBKTLTUABOhä9AcyCOLBsl5Pi

The variables are digits (not characters). There is no way back to the start sequence (first one-way-function). Simultaneous the function calculates the following destination Data:

dividing-constant C(k):	1681
position weighted value (H_k):	6798451
destination value (H_p):	581872623626
total value (H_p+H_k):	581879422077

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	=	148	offset cipher alphabet
Beta	(H_k MOD 169) +1	=	89	offset block key
Gamma	((H_p + code) MOD 196) +1	=	128	offset matrix key
Delta	((H_k + H_p) MOD 155) +code	=	93	dynamic bit series
Theta	(H_k MOD 32) +1	=	20	offset back factor

The above control parameters serve for solution of several cryptographical tasks.

3.6 Contraction to BASIS VARIATION

In order to reduce the variables back to decimal values a contract function is introduced.
The digits of 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 parameter **Theta** is deducted. Results are stored in BASIC VARIATION, an array of 16x16 elements. A backwards searching of foregoing Data is not possible (second one-way-function).

The first five reconversions beginning at variante = **1** shows as follows:

3 digits base 78	decimal	Modulo 256	- Theta	element
DBl	79997	125	20	105
Blx	70649	249	20	229
lxä	290619	59	20	39
xäe	364378	90	20	70
äel	422963	51	20	31

BASIC VARIATION (256 elements)

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

3.7 Calculation of „CypherMatrix“

Calculating CypherMatrix the elements of BASIC VARIATION are used directly in their distribution 16x16 to achieve the destination base. The elements values are related to index values of bytes (**0** to **255:** comparable with ASCII-set). The program includes two modes to distribute the characters: variant **B** and variant **D:**

3.7.1 Swap displacing of indexes: i,j (variant B)

Threefold permutation of the elements to achieve the final distribution in the CypherMatrix:
In parts of source code:

```
N = Alpha - 1
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
                INCR N
                Matrix$(1,i,j)  = VARIATION$(N)          elements
             CASE 2
                Matrix$(2,a,j)  = Matrix$(1,i,j)         swapping
             CASE 3
                Matrix$(3,i,a)  = Matrix$(2,i,j)     final matrix
          END SELECT
       NEXT j
    NEXT i
 NEXT s
```

3.7.2 Dynamic generating of indexes: (variant D)

All index values (16x16) are new generated from the elements of BASIC VARIATION in a special array **IndexFolge(2,16)** (another application of CypherMatric function).
In parts of source code:

```
SHARED IndexFolge(2,16), Delta, Omega

FOR B = 1 TO 2
   IF B=1 THEN X = Delta
   IF B=2 THEN X = Omega
   FOR C = 1 TO 16
      INCR X
      IF X > 256 THEN X = 1
      A = VARIATION(X) MOD 16                elements of BASE VARIATION
      IndexFolge(B,C) = A
      IF C>1 THEN
         L = 0
         DO
           INCR L
           IF IndexFolge(B,L) = A THEN
              INCR A
              A = (A MOD 16)
              IndexFolge(B,C) = A
              L = 0
           END IF
         LOOP UNTIL L = C-1
      END IF
      IndexFolge(B,C)  =  IndexFolge(B,C) + 1    array IndexFolge (2,16)
   NEXT C
NEXT B

N = Alpha
FOR I = 1 TO 16
   FOR J = 1 TO 16
      X = IndexFolge(1,I)
```

```
      Y = IndexFolge(2,J)
      Matrix$(X,Y) = VARIATION$(N)          generation of CypherMatrix
      INCR N
      IF N > 256 THEN N = 1
    NEXT J
NEXT I
```

Final CypherMatrix (Variation: D)

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

3.8 Control Parameter

As cipher alphabet (system alphabet of bitsystem on base 7) 128 characters are taken from the actual CypherMatrix at position **148.** Certain signs (hex: 00 bis 20, 22, 2C, B0, B1, B2, D5, DB, DC, DD, DE, DF and others) are ignored because they do still their original task (e.g. **1A** = ASCII-26) and disturb the proper realization of the program.

Cipher alphabet (base 7)

1	õ	Ò	a	C	y	Đ	k	S	c	ª	N	›	©	¼	.	?	*16*
17	U	÷	[?]	ˆ	Z	è	Ë	g	_	b	f	s	ô	(	+	ð	*32*
33	″	â	¶	°	À	–	¿	`	[?]	d	Ñ	ê	Ó	9	Q	Ù	*48*
49	)	>	‡	µ	K	P	‟	'	½	h	ò	ù	í	¸	Ê	W	*64*
65	È	R	€	œ	e	t	<	á	0	‹	X	Ç	¨	o	H	ä	*80*
81	Ã	-	J	;	«	ö	[	ñ	8	Ì	$	Å	E	ü	,	†	*96*
97	¾	|	6	Ö	q	[?]	¹	[?]	Ž	j	´	T	•	Y	{	l	*112*
113	û	5	]	Í	Î	V	„	:	Š	É	$	é	¡	ï	Â	□	*128*

Cipher alphabet (hex)

1	F5	D2	61	43	79	D0	6B	53	63	AA	4E	9B	A9	BC	2E	3F	*16*
17	55	F7	81	88	5A	E8	CB	67	5F	62	66	73	F4	28	2B	F0	*32*
33	94	E2	B6	BA	C0	96	BF	60	8F	64	D1	EA	D3	39	51	D9	*48*
49	29	3E	87	B5	4B	50	93	91	BD	68	F2	F9	ED	B8	CA	57	*64*
65	C8	52	80	9C	65	74	3C	E1	30	8B	58	C7	A8	6F	48	E4	*80*
81	C3	2D	4A	3B	AB	F6	5B	F1	38	CC	24	C5	45	FC	82	86	*96*
97	BE	7C	36	D6	71	8D	B9	A0	8E	6A	B4	54	95	59	7B	6C	*112*
113	FB	35	5D	CD	CE	56	84	3A	8A	C9	A7	E9	A1	EF	C2	FF	*128*

Block key

The block key taken from the actual CypherMatrix at position **89** comprises 42 characters:

\¢ƒæß#·®L}ÚŒ€F'³' à™AI@%ØÝÁ¯ž#vå‹ý€2ã*Ÿ˜²

```
5C A2 83 E6 DF 23 B7 AE 4C 7D DA 8C 9D 46 27 B3 92 20 E0 99 41
49 40 25 D8 DD C1 AF AD 9E 05 76 E5 CF FD 90 32 E3 2A 9F 98 B2
```

Matrix key

As start sequence for the next round the program takes at position **128** a new matrix key with 42 characters:

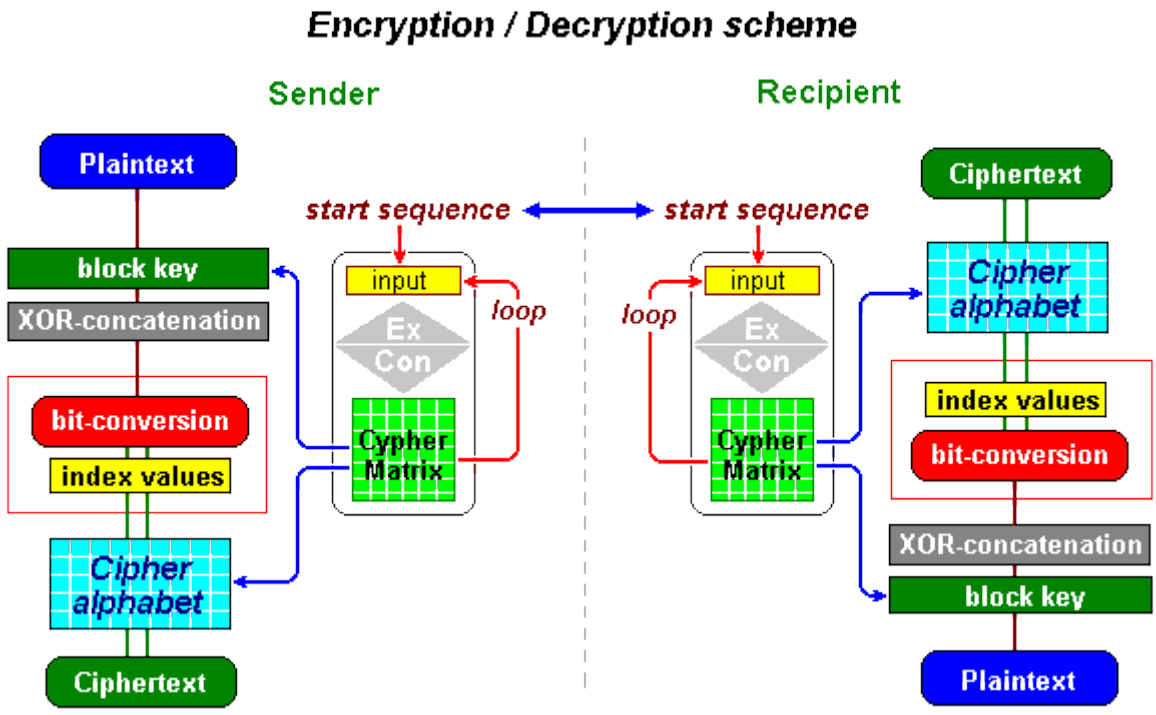
Ÿ˜²^ìç‰zw,ui¬ø3ór—MÔõ#ÒaCyÐkScªN›©#¼.?#U÷€

```
9F 98 B2 5E EC E7 89 7A 77 2C 75 69 AC F8 33 F3 72 97 4D D4 F5
11 D2 61 43 79 D0 6B 53 63 AA 4E 9B A9 12 BC 2E 3F 14 55 F7 81
```

The matrix key is to be lead back to the beginning of the function (**loop**). The respective matrix key controls the entire course of the program in equal way at sender and recipient as well .

4 Coding area

Encryption – writing and reading of secret informations – is exclusively performed in the coding area. By inserting an identical start sequence at sender and recipient es well an equal course and identic control parameters are generated. The following schema shows the connections:



Ciphering is performed by the following alternatives:

1. **Basic Coding:** bit conversion without further operations or
2. **Compound Coding:** bit conversion with additional operations ,
   a) with XOR-concatenation (foregoing or succeding),
   b) connected with further operations (see: "Combinierte Operationen")

Encryptions in CypherMatrix procedure are very simple:

1. The generator creates the necessary **system alphabet** and
2. the cipher text is written by **bit conversion**.

4.1 Program overview

Up to now the Author has developed the following encryption programs:

System Basis	System Alphabet	Basis-Coding (ohne XOR-Funktion)		Verbund-Coding (mit XOR-Funktion)		Längen-verhältnis
		einfache Matrix (m)	dreifache Perm. (p)	einfache Matrix (m)	dreifache Perm. (p)	
1	2	Crypto01	Coding01	MonoCode	MiniCode StepMini	1:8 1:8
2	4	Crypto02	Coding02	ZweiCode	DualCode	1:4
3	8	Crypto03	Coding03	DreiCode	TrialCode	1:2,66
4	16	Crypto04	Coding04	VierCode	FourCode	1:2
5	32	Crypto05	Coding05	QuinCode	FiveCode StepFive	1:1,6 1:1,6
6	64	Crypto06	Coding06	CM64Code	Neu6Code	1:1,33
7	128	Crypto07	Coding7B	DynaCryp DataCryp CodeData[1)] QuadCode[2)]	DynaCode DataCode	1:1,143 1:1,143 1:1,143 1:1,143
8	256	Crypto08 CMCode8D	Coding08	PlanCode	CyphCode StepCyph	1:1 1:1
9	512	Crypto09	Coding09	NeunCode	Cypher89	1:1,79
10	1024	Crypto10	Coding10	ZehnCode	DecaCode	1:1,6
11	2048	Crypt11A Crypt11B	Coding11	ElvaCode	CM11Code StepCM11	1:1,46 1:1,46
12	4096	Crypto12	Coding12	MegaCodA MegaCodB	MaxiCode	1:1,33 1:1,33

[1)] program with three operations (XOR – bit conversion - exchange),
[2)] program with four operations (dyn24 – XOR – bit conversion – exchange).

In addition further programs in bitsystem on base 8 are developed and in bitsystems from base 9 upto base14 as follows:

system base	system alphabet	basic coding (without XOR)	compound coding (with XOR)
8	256	System08.exe	MyCode08.exe
9	512	System09.exe	MyCode09.exe
10	1024	System10.exe	MyCode10.exe
11	2048	System11.exe	MyCode11.exe
12	4096	System12.exe	MyCode12.exe
13	8192	System13.exe	MyCode13.exe
14	16384	System14.exe	MyCode14.exe

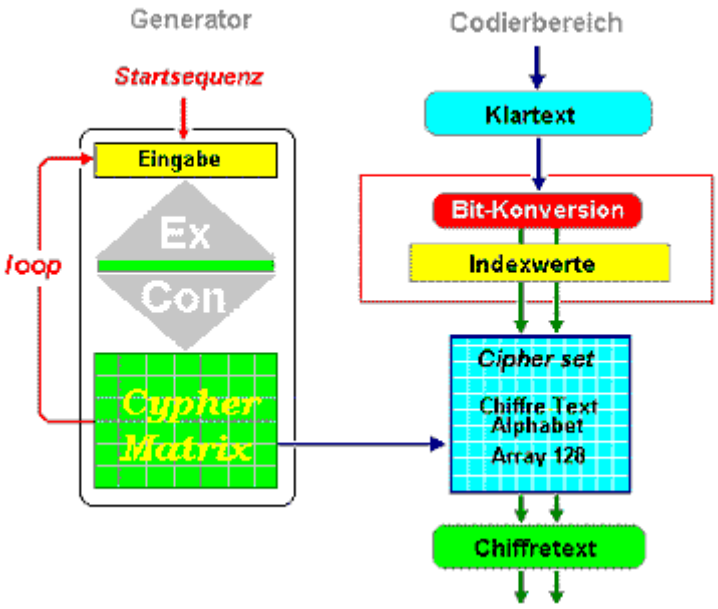
The system-alphabet for the programs in bitsystem on base 8 comprise 256 signs. Because except unicode there are no more single signs available for this system alphabets the digits of **number system** on **base 128** are used as double signs, that are 16384 digits.

The programs are sorted corresponding to their bitsystematics (base 1 to base 14). All programs are DOS-programs and run under WindowXP, only. They have to be converted to **C#,** as has already done for the programs **DataCryp.exe** and **DynaCode.exe** under management of Prof. **Bernhard Esslinger** (Uni Siegen) and his CrypTool team (especially: **Michael B. Schäfer**) [#8]. After converting C#-programs run 25 times faster than the original DOS-programs. All other programs still have to be converted.

4.2 „Basic Coding“

In „**Basic coding“** bit conversion works directly from plaintext in bitsystem on **base 8** to ciphertext in bitsystem on **base 7** (alternativ: to base 1 up to base 12). Inserted plaintext bit series of 42 x 8 bits (336 bits) result to converted bit series of 48 x 7 bits (336 bits). Values in bitsystem on base 7 serve for cipher characters indexes in the respective system alphabet on base 7. This is the simpliest way of digitally encryption. In addition the generator is necessary, only.

From the respeptive system alphabet indexes the cipher characters are derivated which then are combined to the cipher text. Connections show as follows:



Changing from plaintext to ciphertext are processed by two functions:

1. Bit conversion
   8-bit plaintext values → 7 bit index values (0 ...127)
2. Destination of ciphertext
   7-bit index values → cipher alphabet (0...127) → ciphertext.

In present case length of plaintext blocks and length of block keys are defined with 42 characters. The start sequence introduced in paragraph 3.2 remains unchanged:

Checkpoint Charly at Madison Square Garden [42 bytes]

Example for encryption is a text from **John Steinbeck** in file **„Steinbeck.txt“** (578 bytes). It is encrypted by program „**Coding7B.exe**“

Early morning is a time of magic in Cannery Row.
In the gray time after the light has come and before
the sun has risen, the Row seems to hang suspended out
of time in a silvery light. The street lights go out,
and the weeds are a brilliant green. The corrugated
iron of the canneries glows with the pearly lucence
of platinum or old pewter. No automobiles are running
then. The street is silent of progress and business.
And the rush and drag of the waves can be heard as
they splash in amomg the piles of the canneries.

John Steinbeck, Cannery Row, New York 1945

As first plaintext block 42 bytes are inserted:

Early morning is a time of magic in Canner

```
45 61 72 6C 79 20 6D 6F 72 6E 69 6E 67 20 69 73 20 61 20 74 69
6D 65 20 6F 66 20 6D 61 67 69 63 20 69 6E 20 43 61 6E 6E 65 72
```

Plaintext blocks in bitsystem on **base 8** are converted to segments in bitsystem on **base 7.** Decimal values of changed signs (7 bits) serve for positions of cipher characters in the actual cipher alphabet (system alphabet). Indexes have to be added by (+1), because the array alphabet does not recognize the value „**0**“.

Plaintext:

	E	a	r	l	y		m	o	. . . . .
base 8:	**01000101**	**01100001**	**01110010**	**01101100**	**01111001**	**00100000**	**01101101**	**01101111**	**. . . . .**
base 7:	**0100010**	**1011000**	**0101110**	**0100110**	**1100011**	**1100100**	**1000000**	**1101101**	**0110111 1 . . . .**
index:	34	88	46	38	99	100	64	109	55 . . . . .
(+1)	35	89	47	39	100	101	65	110	56 . . . . .
ciphertext:	¶	8	Q	¿	Ö	q	È	Y	`. . . .

¶8Q¿ÖqÈY‘Eo¹ÇhH”KEqkªÒŽj“ÌÀké_ÈY)ÌYËbÒJ{UU•ËÍhX] . . . . . .

hexadecimal

```
B6 38 51 BF D6 71 C8 59 91 45 6F B9 C7 68 48 94 4B 45 71 6B AA D2
8E 6A 93 CC C0 6B E9 5F C8 59 29 CC 59 CB 62 D2 4A 7B 55 55 95 CB
CD 68 58 5D
```

Ciphertext **Steinbeck.ctx** comprises 736 characters in system alphabet on base 7.

4.3 „Compound Coding“

By „Compound coding“ several operations are combined in succession (e.g. XOR concatenation) with bit conversion and further operations (dyn24, exchange).

4.3.1 XOR concatenation

In front of bit conversion is installed **XOR concatenation.** Encryption works in three functions:

1. partial dynamic „one-time-pad“
   plaintextblock → block key → 8-bit XOR concatenation
2. bit conversion
   8-bit XOR concatenation → 7 bit index values (0 ...127)
3. destination of ciphertext
   7-bit index values → cipher alphabet (0...127) → ciphertext.

Moduls are passing the procedue in succession.

```
. . . . . .
CALL  XORGenerator (InputData$,TransData$)
CALL BitConversion (TransData$,OutputData$)
. . . . . .
```

4.3.2 Partial dynamic „one-time-pad“

As example are demonstrated encryption steps for **Steinbeck.txt** by program **Dynacode.exe.**
In each round a plaintext block of 42 bytes (42x8 = 336 bits) is XOR concatenated with a block key of equal length. As first plaintext block the program reads:

Early morning is a time of magic in Canner

```
45 61 72 6C 79 20 6D 6F 72 6E 69 6E 67 20 69 73 20 61 20 74 69
6D 65 20 6F 66 20 6D 61 67 69 63 20 69 6E 20 43 61 6E 6E 65 72
```

Block key taken at position **149** of the actual CypherMatrix comprises:

\¢ƒæß#·®L}ÚŒ€F'³' à™Al@%ØÝÁ¯ž#vå Ïý€2ã*Ÿ˜²

```
5C A2 83 E6 DF 23 B7 AE 4C 7D DA 8C 9D 46 27 B3 92 20 E0 99 41
49 40 25 D8 DD C1 AF AD 9E 05 76 E5 CF FD 90 32 E3 2A 9F 98 B2
```

XOR concatenation:
plaintext:

	E	a	r	l	y		m	o	. . . . .
base 8:	**01000101**	**01100001**	**01110010**	**01101100**	**01111001**	**00100000**	**01101101**	**01101111**	**. . . . .**
block key:									
	01011100	**10100010**	**10000011**	**11100110**	**11011111**	**00100011**	**10110111**	**10101110**	**. . . . .**
XOR:	**00011001**	**11000011**	**11110001**	**10001010**	**10100110**	**00000011**	**11011010**	**11000001**	**. . . . .**
hex:	19	C3	F1	8A	A6	03	DA	C1	
ASCII:	€	Ã	ñ	Š	¦	€	Ú	Á	

€ÃñŠ¦€ÚÁ>#³âúfNÀ²lÀí($%€·»áÂÌùl#Å¦"°q,DñýÀ . . . . . .

```
19 C3 F1 8A A6 03 DA C1 3E 13 B3 E2 FA 66 4E C0 B2 6C C0 ED 28
24 25 05 B7 BB E1 C2 CC F9 6C 15 C5 A6 93 B0 71 82 44 F1 FD C0
```

A partial dynamic „**one-time-pad**“ is achieved. Plaintext and block key are of equal length and the key will not be repeated. Each round gets another key from the respective CypherMatrix.

4.3.3 Bit conversion

Result of XOR concatenation in bitsystem on base 8 (42x8 = 336 bits) is converted to signs of bitsystem on base 7 (336 bits = 48x7). Decimal values of converted signs (base 7) are index values to positions of characters in the cipher alphabet (system alphabet). Indexes for cipher alphabets have to be added with (+1) because the array alphabet does not recognize the value „**0**“.

bit conversion:
XOR base 8:

	00011001	**11000011**	**11110001**	**10001010**	**10100110**	**00000011**	**11011010**	**11000001**	**. . . . .**
base 7:									
	0001100	**1110000**	**1111110**	**0011000**	**1010101**	**0011000**	**0000111**	**1011010**	**1100000** **1 . . . . .**
index:	12	112	126	24	85	24	7	90	96
(+1)	13	113	127	25	86	25	8	91	97
alphabet (base 7)									
	©	û	Â	_	ö	_	S	$	¾

As result of bit conversion we get the ciphertext file **Steinbeck.ctx** (672 bytes) as follows:

©ûÂ_ö_S$¾ä€ùð›Î¹`)Ë¿¹C$□□ª”Å¸lœ€¹Ê9RQËoˆ8ô)À`áéÈfŽKVTn}ÿB5õ¥¶Âûò5éäçfhøªÅ·$ïZB)þÖ□Ï1‘‰ò)m÷eJ~BlÇ˜«¾×gM)□ññÁÿ@2ýr¨}v6.œÖ4ï=v\×M□BûÂÏLÑrÌŒÁ`åÆÇ>iÑÀeŠOØ‡L]Àhv?÷|ýZHôCì3Vi~Ÿ«c¶‡R'€X¢Ì**mTÐýîœg@—□Âì-_“□Çg□j¨? .....

4.4 Threefold compound

In program **CodeData.exe** threefold compound works as follows:

```
. . . . . .
CALL  XORGenerator (InputData$,TransData$)          for „XOR“
CALL  BitConversion (TransData$,DataTrans$)         for „bit conversion“
CALL  ByteWechsel (DataTrans$,OutputData$)          for „exchange“
. . . . . .
```

4.5 Fourfold compound

In program **QuadCode.exe** exists the following assembly:

```
. . . . . .
CALL  Substitution (InputData$,ReportData$)         for „dyn24“
CALL  XORGenerator (ReportData$,TransData$)         for „XOR“
CALL  BitConversion (TransData$,DataTrans$)         for „bit conversion“
CALL  ByteWechsel (DataTrans$,OutputData$)          for „exchange“
```

By the same way other operations may be connected with the respective program. Analysing further programs of the overview in paragraph 4.1 of this article is shown in Ergänzung zum Paradigma in der Kryptographie (sorry, in German language only) . In order to test the procedure you may download the program **DataCryp** with an encrypted file **Message8.ctx** as **Datatest.zip** and try to decipher the messages.

5 Deciphering

For deciphering the generator performes an identic course equal to the encryption process. Deciphering is performed in the coding area, but in reverse order, only.

1. Analysing the ciphertext
   ciphertext → cipher alphabet (0...127) → 7 bit index values
2. bit conversion
   7 bit index values (0 ...127) → 8-bit XOR concatenation
3. XOR concatenation
   8-bit XOR concatenation → block key → plaintext block

The program searches in blocks of **48** cipher characters the decimal index values of single characters in an identical created cipher alphabet (system alphabet) and connects them to a series of 336 bits. This series will be devided into **42** 8-bit sequences (336 bits) in bitsystem on base 8 and concatenated with the respective block key. The original plaintext becomes visible.

6 Security of the procedure

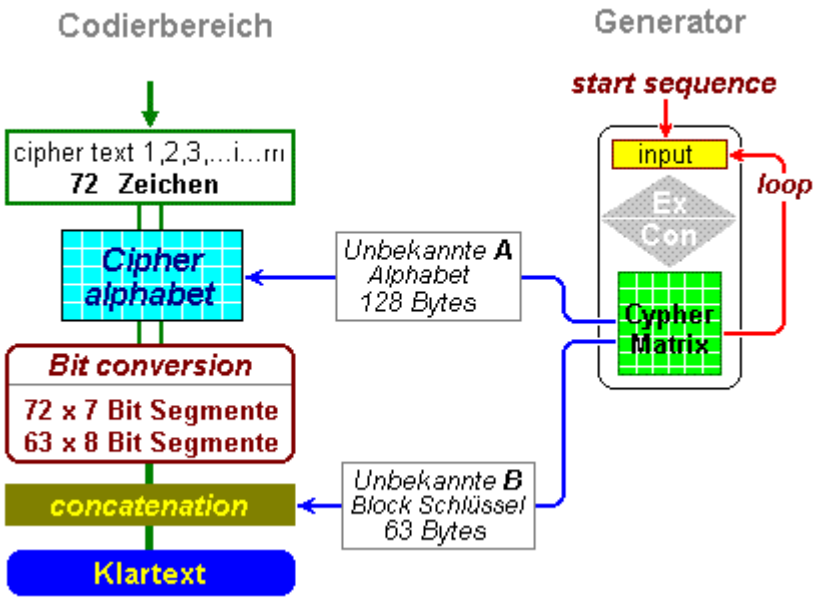
The most known attacks are analysing structures, „known plaintext attack“ and „chosen plaintext attack“, possibly still „differential“ and „linear“ analysis. By this attacks it is intended to acquire statistically repetition patterns and word combinations, frequency structures and two-digit-groups, in order to find a way to the plaintext, possibly [#9].

However, an analysis on basis of this features presupposes that plaintext and ciphertext can be compared with each other. Insofar there must be a unity order system which works in plaintext and ciphertext as well. In almost all present procedures plaintext and ciphertext have the same length: **"Längenkongruenz"** [#9]. According to this fact there must exist for each single plaintext character a specific ciphertext character. Hence, both areas work in the same bitsystem on base 8 with the result that there is a uniform order system and both areas can be compared, too.

Otherwise in encrypting procedures according to CypherMatrix method a changing in bitsystems is achieved. In ciphertext area arises a new order system whith result that both areas cannot be compared any longer. Congruence of length is missing. By changing signs from bitsystem on base 8 to signs in bitsystem on base 7 upon each plaintext character falls a converted ciphertext character which is longer by factor 1,143 (8/7) than the respective plaintext character. Due to the fact both areas cannot be compared with each other the basis fails for almost all conventional attacks. Consequently they are without effect and we may forget them [#10].

At least there is still the possibility of „brute force attack“. Pricipally an attacker knows the ciphertext and the CypherMatrix method with its functions, but he does not know the actual start sequence and following parameters: **alpha, beta** and **gamma.** He only may try to iterate all possibilities. An attack on the start sequence with **42 bytes** results to an entropie of **336** and an exponential complexity of **O(2^336)** = 1.4E-101. The attacker using the ciphertext may then try to find single parts of encryption steps. But there are no starting points which give some expectation for success.

Decryption happens in each round as follows:



The procedure includes three functions:

1. Plaintext block --> **block key** --> -8 bit XOR sequences
2. 8-bit XOR sequences --> 7-bit index values
3. 7-bit index values --> **chipher alphabet (128)** --> ciphertext

In this functions the parameters **block key** and **cipher alphabet** are two variables independent from each other. Effective are:

cm = f [f1 (an, **k1**), f2 (b1, b2), f3 (b2, **k2**)]
an = f [f3 (cm, **k2**), f2 (b2, b1), f1 (b1, **k1**)]

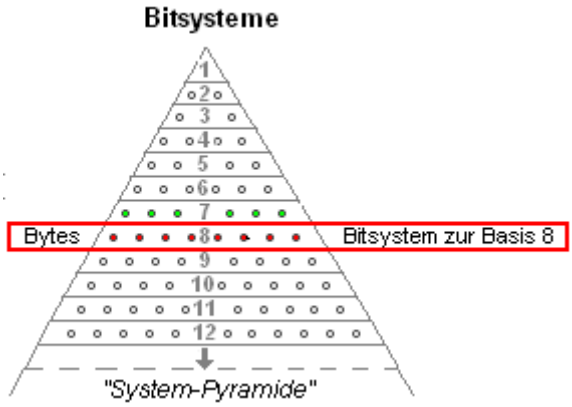
fx = functional connection
an = plaintext
k1 = **block key**
b1 = 8-bit sequence
b2 = 7-bit index value
k2 = **cipher alphabet (128)**
cm = ciphertext

Ascertaining the ciphertext (**cm**) and retrograde searching for the plaintext (**an**) points out as an equation with two unknown veriables: **k1** und **k2.** That only leads to a definite solution, if one unknown variable can be derived from the other unknown variable or if there are two equations with the same unknown variables.

But between the respective block key = k1 and the cipher alphabet (128) = k2 generated in the same round there are no connections. Nevertheless, even if both are extracted out of the current CypherMatrix yet they have no functional connection: (**k1** --> (**Hk MOD 169**)**+1**) and **k2** --> (**Hk +Hp**) **MOD 255**)**+1**). The current CypherMatrix itself is derivated from the initial start sequence only. But thereunto is no way back (two one-way-functions prevent this). Hence, there are many couples of cipher alphabets / block keys which result from an „brute force“ attack and deliver any legible texts, but one does not know which is the right one: Angriff mit "brute force" . Thus „brute force“ cannot have success, too.

7 Summary

In the field of dynamic bit series there are following efficient connections and structures :
Bit series have to be divided in defined segments (length) and ordered in systematics of bit sequences (systemized). As scale their length may be chosen: **bitsystem** on b**ase** of defined **length**.



In order to make operation results visible an independent system alphabet (cipher alphabet) has to be defined. System alphabet for base 1 is the binary system with two bits. At beginn of computer techniques the simple ASCII-system with 128 characters was defined as a sytem alphabet.. When it came clear that this system was to narrow the bitsystem was enlarged to base 8 and to extended ASCII-system with 256 characters. For bitsystem on base 16 there is the system alphabet **Unicode.**

Nowadays is used the bitsystem on base 8, especially for encryption. Mostly plaintext in bitsystem on base 8 is enciphered to ciphertext issued in bitsystem on base 8 as well. This becomes evident by the fact that ciphertext has – or shall have – same length as plaintext [#12], in both cases: **8 bit.** Accordingly no system changing takes place.

In CypherMatrix procedures always is a system changing from bitsystem on base 8 to any other bitsystem, best to bitsystem on base 7. Here ciphertext comprises **1,143** times of plaintext length. Definition of system alphabets and controlling encryption is task of the **generator.** The system alphabet alone arranges the ciphertext. A **key** in traditional sense to integrate into the encryption steps is not necessary. Only for initiation of the generator a start sequence (passphrase) of optimal 42 characters is required. Insofar it seems making more sense to speak of „coding“ instead of „encryption“.

More details of **CypherMatrix** procedures you will find under: www.telecypher.net/
Who wants to deal with the procedures more intensively may request single programs – with or without source code – from the author per e-mail and work with them under the CMLizenz **.**
(eschnoor@multi-matrix.de).

Munich, in August 2012

[#1] Schneier, Bruce, Angewandte Kryptographie (dt. Ausgabe), Bonn … 1996, S.229

[#2] Schmeh, Klaus, Safer Net, Kryptographie im Internet und Intranet, Heidelberg 1998, S. 61
[#3] Paar, Christof und Pelzl, Jan, Understanding Cryptography, Berlin-Heidelberg, 2010, S. 124
[#4] Bräkling, André, www.braekling.de/web-development/6-bits-und-bitfolgen.html
[#5] Morin, Charles, www.kbcafe.com/articles/HowTo.Base64.pdf
[#6] Feistel-Netzwerke, www-lehre.inf.uos.de/~rspier/referat/feist.html
[#7] Strukturvergleich Klartext und Geheimtext, http://www.telecypher.net/Equilang.pdf
[#8] Esslinger, Bernhard, Uni Siegen, http://www.cryptool.org/de/
[#9] Strukturvergleich Klartext und Geheimtext, www.telecypher.net/Equilang.pdf
[#10] Kryptanalyse des Verfahrens, www.telecypher.net/CYPHKERN.HTM#28
[#11] Strukturvergleich Klartext und Geheimtext, www.telecypher.net/Equilang.pdf
[#12] Paar, Christof und Pelzl, Jan, a.a.O., S.75
[#13] Core of the cypherMatrix procedure, /www.telecypher.net/CORECYPH.HTM