MEMO

Paradigmenwechsel in der Kryptographie

Ernst Erich Schnoor

Kryptographie ist **schreiben** und **lesen v**on **geheimen** Botschaften mit arteigenen Methoden (Verschlüsselung). Seit sich die Computer der Kryptographie bemächtigt haben, verwirren **Bits** und **Bytes** die Anwender. Dabei scheint es doch so einfach: 8 Bits sind 1 Byte. Das vermittelt Einheitlichkeit. Aber dennoch lassen sich in der Kryptographie unterschiedliche Techniken entwickeln. Wie eine Untersuchung der Längenverhältnisse zwischen **Klartext** und **Chiffretext** zeigt, sind im Prinzip beide Texte gleich lang (ohne Header, Kontrollsequenzen, Padding, Zeitstempel u.ä.). Sie sollen auch gleich lang sein, damit der Chiffretext wieder an der gleichen Stelle abgelegt werden kann, an der der Klartext gespeichert war [#1].

Wenn aber beide Texte gleich lang sind, folgt daraus zwingend, dass sie auch die gleiche Anzahl Zeichen enthalten **(Bytes)** [#2,#3]. Auf jedes Klartextzeichen entfällt somit ein - und zwar nur **ein -** bestimmtes Chiffretextzeichen. Die Verschlüsselung vom Klartextzeichen zum Chiffretextzeichen geschieht daher allein durch eine **funktionale** Änderung der 8 Bits innerhalb eines Bytes (**Bitfolge**), auch wenn der Weg dahin vielleicht über längere Bitsequenzen geht.. Da sowohl im Klartext als auch im Chiffretext Zeichen mit 8 Bits (Bytes) verwendet werden, muss sich die Änderung in einem einheitlichen System (jeweils 8 Bit) und in einem identischen System-Alphabet von 256 Zeichen vollziehen (bisheriges **Paradigma** der Kryptographie). Aber, die Begrenzung auf Bytes mit 8 Bit ist offensichtlich zu eng. Es fehlen wesentliche reale Vorgänge, die insbesondere in einer **Systematik** der Bitfolgen und den **System-Alphabeten** zur Visualisierung der Informationsinhalte zum Ausdruck kommen.

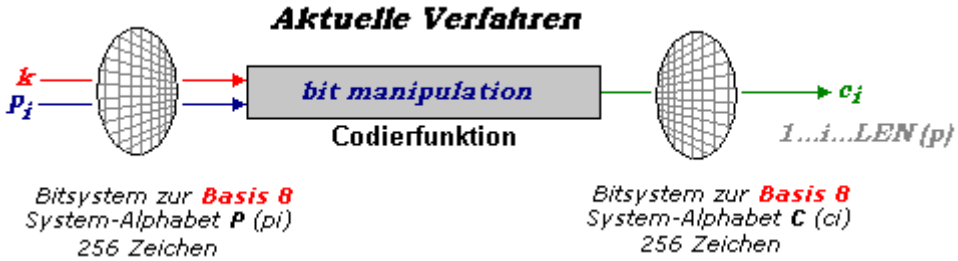
1 Systematisierung der Bitfolgen

Der Computer kann datentechnisch nur Bitfolgen von „0“ und „1“ verarbeiten. Sie sind in ihrer Menge unbegrenzt (1 bis ∞). Um mit Bitfolgen zu arbeiten, Gesetzmäßigkeiten zu erkennen und Programme zu gestalten – insbesondere auch Verschlüsselungen – müssen die Bitfolgen in definierte Abschnitte (**Einheiten**) geteilt , d.h. systematisiert werden. Es liegt ein vergleichbares Phänomen vor, wie bei der Menge aller Zahlen. Wie in der Zahlentheorie lassen sich auch die Bitfolgen in einem Stellenwertsystem ordnen. Dabei entsprechen das Bit der Ziffer, die Einheit (bzw. das Byte) der Zahl und das System-Alphabet stellt die geordnete Menge dar. Für 8-bit Folgen kann man dann beispielsweise vom "**Bitsystem** zur **Basis 8**" sprechen. Im Einzelnen ergeben sich:

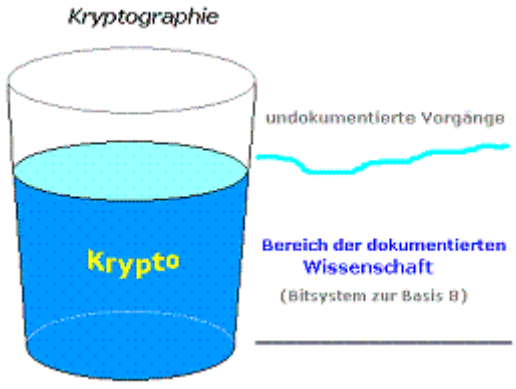
				System-Alphabet
Bitfolgen:	1-bit = Bitsystem zur Basis 1	=	2^1 Zeichen =	2 Units
	2-bit = Bitsystem zur Basis 2	=	2^2 Zeichen =	4 Units
	3-bit = Bitsystem zur Basis 3	=	2^3 Zeichen =	8 Units
	4-bit = Bitsystem zur Basis 4	=	2^4 Zeichen =	16 Units
	5-bit = Bitsystem zur Basis 5	=	2^5 Zeichen =	32 Units
	6-bit = Bitsystem zur Basis 6	=	2^6 Zeichen =	64 Units
	7-bit = Bitsystem zur Basis 7	=	2^7 Zeichen =	128 Units
	8-bit = Bitsystem zur Basis 8	**=**	**2^8 Bytes =**	**256 Bytes**
	9-bit = Bitsystem zur Basis 9	=	2^9 Zeichen =	512 Units
	10-bit = Bitsystem zur Basis 10	=	2^10 Zeichen =	1024 Units
	11-bit = Bitsystem zur Basis 11	=	2^11 Zeichen =	2048 Units
	12-bit = Bitsystem zur Basis 12	=	2^12 Zeichen =	4096 Units
	16-bit = Bitsystem zur Basis 16	=	2^16 Zeichen =	65536 Units
	32-bit = Bitsystem zur Basis 32	=	2^32 Zeichen =	4294967296 Units

Bisher sind in der Kryptographie nur folgende Bitfolgen als Einheiten definiert worden: Das Bitsystem zur Basis 1 umfasst die Zeichen: „0“ und „1“ oder historisch: „L“ und „H“. Das Bitsystem zur Basis 4 kennt die Einheit „nibble“ und 8-bit Sequenzen die Einheit: „**Byte**“. Im Bitsystem zur Basis 16 gibt es die Einheiten: “word“ ,“dword“. Alle anderen Bitfolgen werden jeweils mit der enthaltenen Anzahl der Bits gekennzeichnet [#4].

Mit Einführung der digitalen Technik haben die zur Weiterentwicklung der Kryptographie berufenen Wissenschaftler sich vordergründig auf das Bitsystem zur Basis 8 gestützt und dabei offensichtlich eine sachgerechte **Systematisierung** der Bitfolgen als nachrangig angesehen. Fast alle Operationen vollziehen sich im Bitsystem zur **Basis 8.** Sowohl Eingaben werden im Bitsystem zur Basis 8 mit dem System-Alphabet von 256 Zeichen (00 bis FF) verarbeitet als auch verschlüsselte Ergebnisse (Chiffretext) im Bitsystem zur Basis 8 und System-Alphabet mit 256 Zeichen ausgegeben. Dass zwischen Eingabe und Ausgabe vielfach Bitfolgen anderer Länge abgearbeitet werden (Codierfunktion), ist für das Ordnungsprinzip nicht entscheidend. Insoweit vollziehen sich alle Verschlüsselungsoperationen – unabhängig von der Anzahl der zwischenzeitlich manipulierten Bits – in einem einheitlichen **Ordnungsystem**, im Bitsystem zur **Basis 8**.



Mit den geschilderten Zusammenhängen und der Forderung, Klartext und Chiffretext müssten gleich lang sein, wird der **Aktionsraum** der Kryptographie auf den Bereich des Bitsystems zur **Basis 8** mit dem System-Alphabet (Chiffre-Alphabet) von **256 Bytes** begrenzt. Damit bleiben über das Bitsystem zur Basis 8 hinaus wesentliche Zusammenhänge und Möglichkeiten außer Betracht.. An dieser Stelle zeigen die wissenschaftlichen Analysen der Kryptographie offensichtlich einige undokumentierte Bereiche, gewissermaßen ein „**crypto incognito**“.



Mit der Systematisierung der Bitfolgen und dem jeweils zugeordneten System-Alphabet wird eine neue Technik der Verschlüsselung sichtbar. Es eröffnen sich eine Vielzahl neuer Möglichkeiten, sowohl für die Praxis als auch für die Forschung (neues **Paradigma** der Verschlüsselung).

2 Bit-Konversion

Bit-Konversion ist die Umwandlung einer Bitfolge von einem Bitsystem in ein anderes Bitsystem. Dabei bleiben die Anzahl der Bits und ihre Reihenfolge gleich. Kein Bit wird hinzugefügt und kein Bit wird weggelassen. Nur die Struktur wird verändert. Die dezimalen Werte der neuen Einheiten sind Indexwerte für die Zeichen des zugeordneten System-Alphabets. Eine Bit-Konversion kann für alle Bitsysteme von zur Basis 1 bis zur Basis 12 (und höher) durchgeführt werden.

Die zugrunde liegenden Zusammenhänge zeigt die folgende Übersicht:

Systembasis	System-alphabet	Bitfolgen in Einheiten Indizierung	Längen-verhältnis
Bitsystem zur Basis 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 zur Basis 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 zur basis 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 zur Basis 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 zur Basis 5	32	01001 11001 10111 10111 01000 11000 01011 00010 01.. 9 25 23 23 8 24 11 2	1:1,6
Bitsystem zur Basis 6	64	010011 100110 111101 110100 011000 010110 001001 ... 19 38 61 52 24 22 9	1:1,33
Bitsystem zur Basis 7	128	0100111 0011011 1101110 1000110 0001011 0001001 ... 39 27 110 70 11 9	1:1,143
Bitsystem zur Basis 8 *Bereiche*	256	01001110 01101111 01110100 01100001 01100010 011.. 78 111 116 97 98 4E 6F 74 61 62 N o t a b ... **Stromchiffre, Blockchiffre mit Längenkongruenz, Feistel-Netze ECB, CBC, CFB, OFB, DES, IDEA, AES, RSA, differenzielle und lineare Kryptanalyse, fast alle weiteren Programme**	1:1
Bitsystem zur Basis 9	512	010011100 110111101 110100011 000010110 001001100 156 445 419 22 76	1:1,79
Bitsystem zut Basis 10	1024	0100111001 1011110111 0100011000 0101100010 01100 313 759 280 354	1:1,6
Bitsystem zur Basis 11	2048	01001110011 01111011101 00011000010 11000100110 ... 627 989 194 1574	1:1,46
Bitsystem zur Basis 12	4096	010011100110 111101110100 011000010110 001001100 1 1254 3956 1558 613	1:1,33
Bitsystem zur Basis xx	*div*	x x x x x	1: xxx

Bisher werden Umwandlungen von Bitfolgen nur im Verfahren „**Coding Base64“** vorgenommen [#5]. Dabei werden Bytes im Bitsystem zur Basis 8 in eine Folge von 6-bit Sequenzen umgewandelt. Die dezimalen Werte dieser Sequenzen sind Indizes für ein Chiffre-Alphabet von 64 Zeichen. Das Chiffre-Alphabet (System-Alphabet) wird statisch vorgegeben.

Nach den vorstehenden Grundsätzen ist das CypherMatrix Verfahren entwickelt (Bezeichnung vom Autor) [#13], das im weiteren Verlauf dieser Darlegungen noch ausführlich vorgestellt und erläutert wird. Kern des Verfahrens ist eine Verschlüsselung durch Konversion der Zeichen von einem Bitsystem in ein anderes und Generierung der zugehörigen System-Alphabete durch einen separaten Generator. Eine Reihe von Programmen (67) sind nach diesen Grundsätzen entwickelt worden. Sie sind in Abschnitt 4.1 dieses Artikels (Programmübersicht) einzeln aufgeführt.

2.1 System-Alphabet

Das System-Alphabet ist der wichtigste Bestandteil der Computertechnik, es ist die Grundlage für die Visualisisierung des Inhalts der Bitfolgen. Ohne die sachgerechte Definition eines Alphabets könnte mit dem Computer gar nicht gearbeitet werden.

Das grundlegende System-Alphabet ist der ASCII-Zeichensatz in seiner jeweiligen Ausprägung. Für umfangreichere Zeichensätze – insbesondere fremde Sprachen – wird **Unicode** verwendet. Für jede Ausgabe des Computers ist ein betreffendes System-Alphabet (Array) erforderlich. So zum Beispiel für Bilder (Pixel, graphische Programme), Töne (digital/analog Wandler, Audio-Dateien), digitale Messgeräte, Strich-Code und für alle weiteren Ausgaben von skalierbaren und in Bitsystemen strukturierten Phänomenen.

2.2 Konversion von Basis 1 nach Basis 8

Die Kodierung einer Bitfolge in Abschnitte von anfangs (historisch) 7, dann erweitert 8 Bit, Indizierung mit den Dezimalziffern 0 bis 127 bzw. 0 bis bis 255 und Definition jeweils eines eigenen System-Alphabets (verschiedene ASCII-Zeichensätze mit je 256 Zeichen) stellt im Grunde genommen bereits eine Verschlüsselung dar. Allerdings wird sie kaum als solche wahrgenommen, sondern im Gegenteil als Grundlage der datentechnischen Kommunikation gesehen. Im Einzelnen geschieht die Konversion vom Bitsystem zur Basis 1 zum Bitsystem zur Basis 8 wie folgt:

System-Alphabet: dezimal hexadezimal

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

Die folgende Bitfolge wird umgewandelt:

Bitfolge Basis 1:
01100010011010010111010001100110011011111011011000110011101100101011011110
Bitfolge Basis 8:
01100010 01101001 01110100 01100110 01101111 01101100 01100111 01100101 01101110
Index: 98 105 116 102 111 108 103 101 110

System-Alphabet (ASCII-Zeichensatz):

b	i	t	f	o	l	g	e	n

Index (hexadezimal) = System-Alphabet (hexadezimal):

62	69	74	66	6F	6C	67	65	6E

In der Klartextausgabe lautet die Bitfolge: „**bitfolgen**“
Die hexadezimale Ausgabe lautet: „**626974666F6C67656E**“

Die zugrunde liegende Bitfolge im Bitsystem zur Basis 1 bleibt unverändert. Das Beispiel verdeutlicht, das nur das System-Alphabet gewechselt werden muss, während die nicht strukturierte Bitfolge (Anzahl und Reihenfolge) bei jeder Konversion grundsätzlich gleich bleibt. Auf diese Weise werden alle Konversionen vollzogen.

2.3 Konversion von Basis 8 nach Basis 7

Im folgenden Beispiel wird eine **Bit-Konversion** von Basis 8 nach Basis 7 erläutert.

Bitfolge Basis 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
Bitfolge Basis 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

Chiffre-Alphabet:
œ Ø U G : é ~ † t
Die Bitfolge wird im Bitsystem zur Basis 7 wie folgt dargestellt: **œ ØUG:é~†t**
So kann auf einfache Weise verschlüsselt (besser kodiert) werden. Aus 9 Klartextzeichen werden 10 Chiffretextzeichen (Längenverhältnis: 1:1,143).

Das System-Alphabet zur Basis 7 mit 128 Units ist mit dem Generator des CypherMatrix Verfahrens und der Startsequenz: „Der schwarze Kater fängt immer graue Mäuse“ erzeugt worden. Die Indexwerte sind um (+1) erhöht, da das System-Array den Index „0“ nicht erkennt.

Chiffre-Alphabet (Basis 7)

1	ì ¾ | B º € Ó ¨ a “ Ê ú … Ä ¤ >	*16*
17	ô @ s / õ — › c d Œ 8 *ƒ* ü x •	*32*
33	g h i 6 ¦ z 7 [F „ µ – Ø ž	*48*
49	· œ † G ¡ © » ª % « 9 ¬ A : ¯ &	*64*
65	; ¼ < D H ¿ U ï I À Á J ã Ù æ O	*80*
81	P Â Ã Å Æ Ë Ö × é ê ë t W ö Í]	*96*
97	0 à C j á N 2 ~ Ô ` Q þ ÷ î k Î	*112*
113	X I = ð * ¸ ” Ÿ E u ¥ É ¢	*128*

Chiffre-Alphabet (hex)

1	EC	BE	7C	42	BA	80	D3	A8	61	93	CA	FA	85	C4	A4	3E	*16*
17	F4	40	73	2F	F5	97	9B	63	64	8C	81	38	83	FC	78	95	*32*
33	67	68	69	9D	36	A0	A6	7A	37	5B	46	84	B5	96	D8	9E	*48*
49	B7	9C	86	47	A1	A9	BB	AA	25	AB	39	AC	41	3A	AF	26	*64*
65	3B	BC	3C	44	48	BF	55	EF	49	C0	C1	4A	E3	D9	E6	4F	*80*
81	50	C2	C3	C5	C6	CB	D6	D7	E9	EA	EB	74	57	F6	CD	5D	*96*
97	30	E0	43	6A	E1	4E	32	7E	D4	60	51	FE	F7	EE	6B	CE	*112*
113	58	6C	3D	F0	2A	AD	B8	8D	94	9F	FF	45	75	A5	C9	A2	*128*

2.4 Konversion von Basis 8 nach Basis 12

So kann beispielsweise auch vom Bitsystem zur Basis 8 in das Bitsystem zur Basis 12 verschlüsselt werden. Dabei bleibt die Anzahl der Bits und ihre Reihenfolge gleich. Es wird kein Bit weggelassen und kein Bit hinzugefügt. Nur die Abstände der Einheiten ändern sich. Aus 8-bit Sequenzen werden 12-bit Abschnitte. Die dezimalen Werte der neuen Einheiten sind die Indexwerte für die Zeichen im System-Alphabet zur Basis 12. Das System-Alphabet wird vom Generator des CypherMatrix Verfahrens erzeugt. Es erfordert im vorliegenden Fall 4096 Zeichen.

In Ermangelung direkter Zeichen werden die Ziffern des Zahlensystems zur Basis **64** (4096 Elemente) im System-Alphabet als Doppelzeichen verwendet.

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

Die Bit-Konversion von Basis 8 nach Basis 12 geschieht wie folgt:

Zeichen Basis 8:

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

Bitfolge Basis 8:
01100010 01101001 01110100 01100110 01101111 01101100 01100111 01100101 01101110
Bitfolge Basis 12:
011000100110 100101110100 011001100110 111101101100 011001110110 010101101110

Index:						
	1574	2420	1638	3948	1654	1390
System-Alphabet:						
	gc	TO	fc	5W	fM	jU

Die „bitfolgen“ zeigen sich im Bitsystem zur Basis 12 wie folgt: „**gcTOfc5WfMjU**“
Aus 9 Klartextzeichen werden 6 Doppelzeichen, mithin 12 Chiffretextzeichen (12/9 = 1,33 fach).

2.5 Folgen der Bit-Konversion

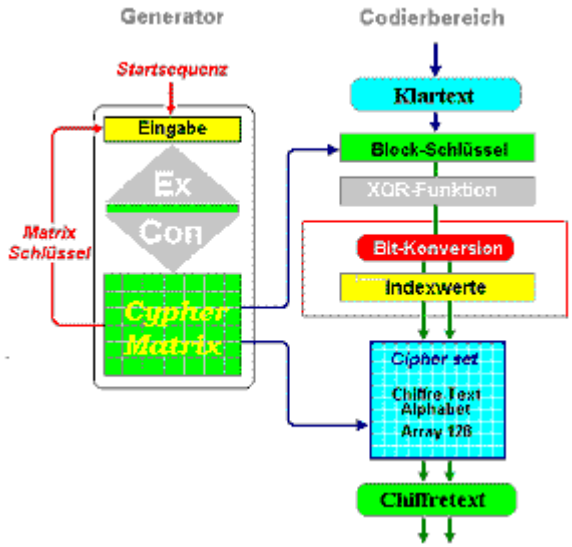
Als Folge der Bit-Konversion zeigt sich eine grundsätzliche Wirkung des Verfahrens: Der Chiffretext wird länger im Verhältnis der Länge der Zeichen im Bitsystem des Chiffretexts zur Länge der Zeichen im Bitsystem des Klartextes. Die Forderung Klartext und Chiffretext sollten gleiche Länge haben, wird im Prinzip nicht erfüllt.
Infolge der Bit-Konversion besteht zwischen Klartext und Chiffretext kein einheitliches Ordnungssystem . Durch die Umwandlung vom Bitsystem zur Basis 8 in z.B. das Bitsystem zur Basis 7 entällt auf jedes Klartextzeichen ein um den Faktor **1,143** längeres Chiffrezeichen. Eine Kryptanalse anhand bestimmter Merkmale, wie beispielsweise Wiederholungsmuster, Häufigkeitsstrukturen, Bigramme, differenzielle und lineare Verfahren [#12] - setzt allerdings voraus, dass Klartext und Chiffretext zueinander im Verhältnis 1:1 stehen. Anders ließen sich wegen fehlender Längenkongruenz auch keine Vergleiche durchführen. Die hierauf gründenden Angriffe können mithin auch keinen Erfolg mehr versprechen.

Den bisher für Verschlüsselungen verwendeten Schritten, wie beispielsw: Feistel-Netzwerke [#6], Spaltenversion, S-Boxen, Betriebsarten ECB, CBC, CFB und OFB, Konfusion und Diffusion und weiteren Prozeduren fehlen im CypherMatrix Verfahren die Voraussetzungen. Es sind kein einheitliches Ordnungssystem und keine Längenkongruenz vorhanden. Die aufgeführten Verschlüsselungsschritte können somit im CypherMatrix Verfahren nicht mehr angewendet werden..

3 Verschlüsselungen mit „CypherMatrix“

Im Gegensatz zu den heute üblicherweise verwendeten Algorithmen geht CypherMatrix eigene Wege. Ein „**Generator**“ erzeugt die zur Durchführung der Verschlüsselung erforderlichen Parameter.

Die Verschlüsselung vollzieht sich in einem getrennten „**Codierbereich**“.



Beide Bereiche werden kombiniert, können aber auch getrennt verwendet werden. Aufgabe des Generators ist die zur Verschlüsselung erforderlichen Steuerungsparameter bereitzustellen, insbesondere das zugeordnete “System-Alphabet”. Die eigentliche Verschlüsselung findet im Codierbereich statt.

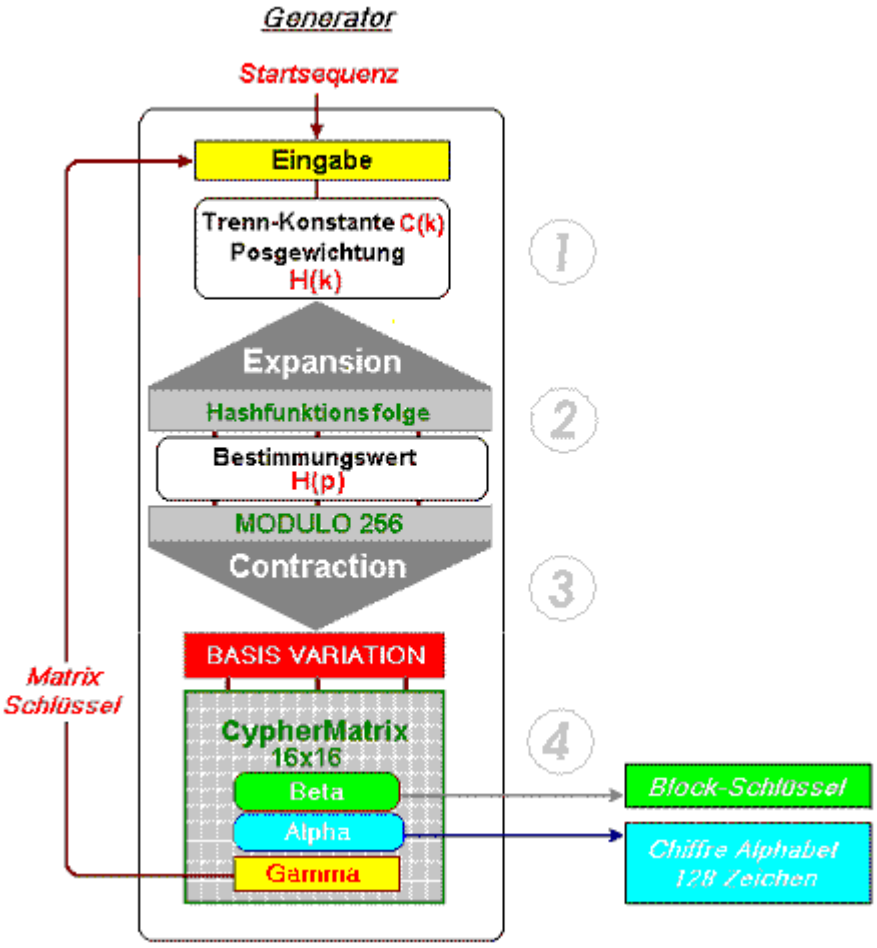
3.1 Der Generator

Das Verfahren ist als Block-Chiffre gestaltet. Es ist symmetrisch, weil Sender und Empfänger zur Initialisierung des Generators die gleiche Startsequenz eingeben müssen und es ist dynamisch, weil der Generator für jeden Klartextblock (zB 63 Bytes) in jeder Runde neue Steuerungsparameter erzeugt. Eine beliebige Startsequenz (Passphrase) mit mindestens 36 Zeichen (optimal 42) steuert das gesamte Verfahren. Einige Beispiele:

Ein Fliegenpilz steigt in die Stratosphäre	[42 Bytes]
Die weiße Elster fließt in das schwarze Meer	[44 Bytes]
Leonardo eroberte Florenz mit Schneekanonen	[43 Bytes]
7 kangaroos jumping along the Times Square	[43 Bytes]

Die Startsequenz sollte ungewöhnlich sein und dennoch leicht zu behalten, so dass sie nicht aufgeschrieben werden muss aber auch nicht geraten werden kann. Wegen ihrer Länge kann sie weder durch Iteration noch durch Wörterbuchangriffe analysiert werden. Ein Angreifer kann auch nicht mit Erfolg versuchen, Teile des Schlüssels getrennt oder nacheinander zu brechen, da die Startsequenz nur in einem Durchgang als Ganzes gefunden werden kann, wenn überhaupt.

Jede Startsequenz erzeugt sowohl beim Sender als auch beim Empfänger einen identischen Ablauf des Verfahrens. Es werden die gleichen Steuerungsparameter erzeugt.



In jedem Durchlauf erzeugt der Generator die zur Verschlüsselung erforderlichen Runden-Parameter:

1. Das Chiffre-Alphabet (System-Alphabet) für die aktuelle Runde,
2. den Blockschlüssel für die XOR-Verknüpfung und
3. einen Matrix-Schlüssel als Startsequenz für die nächste Runde.

Den Abschluss jeder Runde bildet eine **CypherMatrix** mit 16x16 Elementen, die alle Steuerungsparameter für die Verschlüsselung zur Verfügung stellt. Der Matrix-Schlüssel (42 Zeichen) wird auf den Anfang der Funktion zurückgeführt. Er initialisiert den nächsten Durchgang. So entsteht eine unbegrenzte Anzahl von Runden, bis ein Endeimpuls gesetzt wird. Nach der Wahrscheinlichkeit entsteht die Wiederholung einer gleichen CypherMatrix erst in **256!** (Fakultät) = **8E+506** Fällen.

3.2 Eingabe der Startsequenz

Als Beispiel wird die folgende Startsequenz (Eingabe) gewählt:

Der schwarze Kater fängt immer graue Mäuse (n = 42).

Es gilt eine eindeutige Abbildung der Eingabe als Bestimmungsbasis für die Analyse zu finden.

Die Eingabe **m** ist eine Folge bestimmter Bytes **a**(**i**) mit der Länge **n.** Um die Folge als Sachverhalt zu analysieren, muss sie systematisiert (skaliert) werden. Dazu wird jedem Byte **a**(**i**) ein Index zugeordnet und alle **n** Bytes werden in sachgerechter Weise miteinander verknüpft (Addition):

$$\mathbf{m} = \mathbf{a}_1 + \mathbf{a}_2 + \mathbf{a}_3 + \ldots \mathbf{a}_i + \ldots \mathbf{a}_n$$

(Der einzelne Wert für "a_i" wird um (+1) erhöht da sonst ASCII-null (0) nicht berücksichtigt wird)

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

$$\mathbf{m} = 4066$$

Um die einzelnen Bytes **a**(**i**) innerhalb der Zeichenfolge zu unterscheiden, müssen weitere Merkmale hinzukommen, da anderenfalls keine eindeutigen Ergebnisse erzielt werden können.

3.3 Erweiterung zur Positionsgewichtung

Mit Besinnung auf **Renè Descartes** (1596 – 1650) wissen wir, dass jeder Sachverhalt, soweit er in seinen Dimensionen skalierbar ist, durch seine Koordinaten für **Gegenstand, Ort** und **Zeit** (analog kartesischem Koordinatensystem) eindeutig bestimmt werden kann. Die Skalierung erfasst den Gegenstand, den Ort und die Zeit der digitalen Zeichen. Wir definieren:

Sachverhalt = **m** digitale Zeichenfolge der Länge **n**
Gegenstand = **a**(**i**) Element der Folge, Zeichen, Byte
Ort = **p**(**i**) Position von **a**(**i**) innerhalb der Folge
Zeit = **t** (**i**) Zeitpunkt von **a**(**i**) innerhalb der Folge

Damit sich die einzelnen Zeichen unterscheiden wird jedes Byte **a**(**i**) mit seinem Ort **p**(**i**) multipliziert, d.h. **positionsgewichtet**. Die Zeit ist nur dann von Bedeutung, wenn zwischen den einzelnen Bytes und der Prozessorfrequenz eine variable Funktion besteht, ansonsten: **t** (**i**) = **1**. Um einen bestimmten Wert für die Folge **m** zu erhalten, werden die Dimensionswerte für Gegenstand, Ort und Zeit durch Multiplikation verknüpft und zum Zwischenwert **H**(**k**) addiert.

$$\mathbf{H(k)} = \sum_{i=1}^{n} (\mathbf{a}_i + 1) * \mathbf{p}_i * \mathbf{t}_i \qquad \mathbf{t}_i = 1$$

$$\mathbf{H(k)} = 88446$$

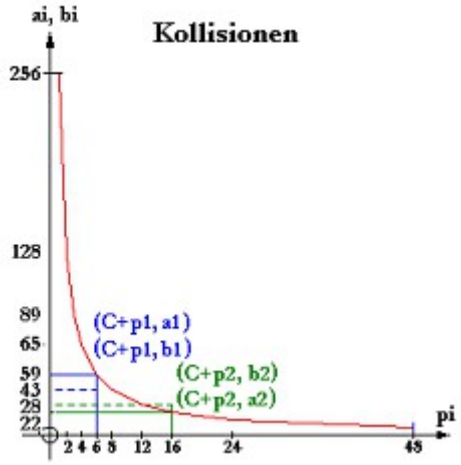
Mit der **Positionsgewichtung** unterscheiden sich zwar die Bytes **a(i),** aber Kollisionen als Folge des Austausches von Bytes innerhalb der Zeichenfolge sind noch nicht ausgeschlossen.

3.4 Ausschluss von Kollisionen

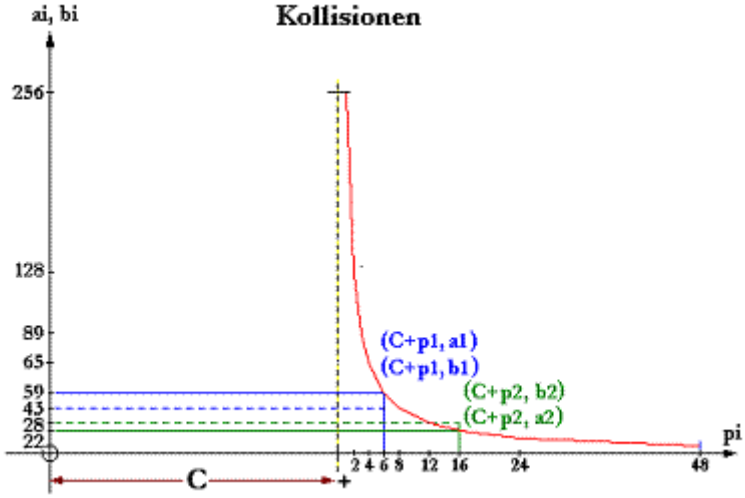
Eine Kollision entsteht unter folgenden Bedingungen:

$$\text{Kollision:} \quad \mathbf{H(k)}\ \mathbf{a}_i = \mathbf{H(k)}\ \mathbf{b}_i$$
$$(\mathbf{a}_1+1) * \mathbf{p}_1 + (\mathbf{a}_2+1) * \mathbf{p}_2 = (\mathbf{b}_1+1) * \mathbf{p}_1 + (\mathbf{b}_2+1) * \mathbf{p}_2$$

An der Stelle $\mathbf{p}_1$ wird das Zeichen $\mathbf{a}_1$ mit dem Zeichen $\mathbf{b}_1$ und an der Position $\mathbf{p}_2$ das Zeichen $\mathbf{a}_2$ mit $\mathbf{b}_2$ ausgetauscht. Die folgende Kurve zeigt die Zusammenhänge zwischen den einzelnen Zeichen $\mathbf{a}_i$ und $\mathbf{p}_i$ im Produkt $(\mathbf{a}_i+1) * \mathbf{p}_i$. :



Um Kollisionen zu vermeiden, wird die Positionsgewichtung in einen Bereich oberhalb der Länge (n) verschoben, d.h. p(i) wird um einen konstanten Abstand C erweitert.



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

Nach Umformung ergibt sich der folgende Veränderungsquotient: **Q**

$$\mathbf{Q} = \frac{(\mathbf{C} + \mathbf{p}_1)}{(\mathbf{C} + \mathbf{p}_2)} = \frac{(\mathbf{b}_2 - \mathbf{a}_2)}{(\mathbf{a}_1 - \mathbf{b}_1)}$$

Für den „Veränderungsquotienten“ - hier mit **Q** bezeichent – sind drei Fälle möglich:

Q > 1
Q = 1
Q < 1

Wenn **Q = 1,** dann müssen auch (**C** + $\mathbf{p}_1$) und (**C** + $\mathbf{p}_2$) gleich sein. Da der Austausch an derselben Position **p** geschieht, ist hier eine Kollision ausgeschlossen. Ist **Q > 1** oder **Q < 1,** dann sind auch ($\mathbf{b}_2 - \mathbf{a}_2$) und ($\mathbf{a}_1 - \mathbf{b}_1$) verschieden. Da die Werte ($\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{b}_1$ und $\mathbf{b}_2$) Integerwerte sind, sind auch deren Differenzen ganzzahlig.

Die Positionen $\mathbf{p}_i$ in der Eingabesequenz mit **N** Bytes (Länge **n**) umfassen einen Bereich von **1** (*minimum*) bis **N** (*maximum*). Der Veränderungsquotient nach der obigen Formel erfasst daher die folgende Spanne:

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

Die weitere Entwicklung ist im Artikel

"Bestimmungsfaktoren für Kollisionsfreiheit"

ausführlich dargelegt. Das Ergebnis führt zur Formel: $\mathbf{C = N * (N - 2)}$

Der Faktor **C** ist allein von der Länge **N** der Eingabesequenz abhängig. Er hat außerdem die Eigenschaften, für gleiche Längen der Eingabesequenz gleich zu sein, und die Zeichen der Positionsgewichtung in kollisionsfreie und kollisionsbelastete Abschnitte zu trennen. Der Faktor **C** erhält daher die Bezeichnung: Trenn-Konstante **C(k)**.

Um die Funktion zu individualisieren wird zusätzlich ein Code – eine gewählte Zahl zwischen 1 und 99 – eingeführt. Wir setzen Code = 1.

$$\mathbf{C(k) = n * (n - 2) + code}$$
$$\mathbf{C(k)} = 1681$$

Nach Einbindung der Trenn-Konstante **C(k)** wird der Zwischenwert $\mathbf{H_k}$ wie folgt ermittelt:

$$\mathbf{H_k} = \sum_{i=1}^{n} \mathbf{(a_i + 1) * (p_i + C_k + Runde)}$$
$$\mathbf{H_k} = 6927458$$

Damit vermeidet das Ergebnis **H(k)** Kollisionen, ist aber immer noch zu niedrig, um unangreifbare Bestimmungswerte für die Funktion zu begründen. Es könnte lediglich als **MAC** für Nachrichten dienen.

3.5 Erweiterung zur Hashfunktionsfolge

Zur Erweiterung der Bestimmungsbasis wird die **Hashfunktionsfolge** (**HF)** eingeführt, die die Eingangssequenz zu einer umfangreichen Folge in einem höherwertigen Zahlensystem expandiert. Das Zahlensystem der Expansion ist wählbar zwischen 64 bis 96. Hier wird die **Basis 77** festgelegt. Für jedes Zeichen der Eingabeseqzenz errechnet das Verfahren den dezimalen Wert ($\mathbf{s_i}$), der dann zu (**di**) **-** Ziffern im Zahlensystem zur Basis 77 - umgewandelt wird. Gleichzeitig ermittelt das Verfahren die Summe aller Einzelergebnisse ($\mathbf{s_i}$) als zusätzlichen Wert **H(p)** zur Bestimmung verschiedener Steuerungsparameter und akkumuliert die Ergebnisse ($\mathbf{d_i}$) seriell zur Hashfunktionsfolge (HF).

$$\mathbf{s_i = (a_i + 1) * p_i * H_k + p_i + code + Runde}$$
$$\mathbf{s_i \rightarrow d_i} \text{ (base 77)}$$

$$\mathbf{HF = d_1 + d_2 + d_3 + \ldots + d_i + \ldots d_m}$$
(m = Anzahl der Zahlen im System zur Basis 77)

$$\mathbf{H_p} = \sum_{i=1}^{n} \mathbf{S_i}$$
$$\mathbf{H_p} = 612705951255$$

Das gewählte Zahlensystem zur Basis 77 umfasst die folgenden Ziffern:

0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz&#@àáâãäåæçèéêë
(definiert vom Autor, nicht standardisiert)

Bei der Generierung der Hashfunktionsfolge ergibt sich beispielsweise für die Teilsequenz "**Kater**" an den Positionen 14 bis 18 der Eingabesequenz folgende Berechnung:

char		pi		Hk	(ai+1)*pi*Hk		Si	Basis 77
	(ai+1)		(ai+1)*pi			pi+code+r		
K	76	14	1064	6927458	7370815312	16	7370815328	2tqFEU
a	98	15	1470	6927458	10183363260	17	10183363277	3wqáiA
t	117	16	1872	6927458	12968201376	18	12968201394	4yä@Qæ
e	102	17	1734	6927458	12012212172	19	12012212191	4Xs&&u
r	115	18	2070	6927458	14339838060	20	14339838080	5MæN2T
						Summe:	612705951255	2çRprTD

Die Hashfunktionsfolge **HF** umfasst 248 Ziffern im Zahlensystem zur Basis 77:

Dk0xBeFd7Vâë3@jQ0ê9S1bMèVh1fIXZO1â@æyi2ZE7Pp2J&dçr2çmCzH3Zmcbk3AG3iM17
fpnm2tqFEU3wqáiA4yä@Qæ4Xs&&u5MæN2T1khAPA5Kèh0f7BVAë26JI3uk69TQlK7ERw1â
28ik0n748s7i7kLèG&7âé7bU7hâáTë8#âcTl2lk4rB8d@KMH9ságHD8elèäFAhâá2à9UmA
t&39leSo7j7ä&2DLD@VDC6BgaJCDI&èzAéHp0ç

Die Variablen sind Ziffern (keine Zeichen) im Zahlensystem zur Basis 77. Es gibt keinen Weg zurück zur Startsequenz (erste Einweg-Funktion). Gleichzeitig errechnet das Programm die folgenden Bestimmungsfaktoren:

Trenn-Konstante C(k):	1681
Positionsgewichteter Wert (H_k):	6927458
Bestimmungswert (H_p):	612705951255
Gesamtwert (H_p+H_k):	612712878713

Aus den Bestimmungsfaktoren werden folgende Steuerungsparameter abgeleitet:

Variante	(H_k MOD 11) +1	=	11	Beginn der Kontraktion
Alpha	((H_k + H_p) MOD 255) +1	=	204	Offset Chiffre-Alphabet
Beta	(H_k MOD 169) +1	=	149	Offset Block-Schlüssel
Gamma	((H_p + code) MOD 196) +1	=	141	Offset Matrix-Schlüssel
Delta	((H_k + H_p) MOD 155) +code	=	44	dynamische Bitfolgen
Theta	(H_k MOD 32) +1	=	3	Offset Rückrechnung
Omega	(H_k MOD 95) +1	=	59	Beginn Doppelzeichen

Die Steuerungsparameter dienen zur Lösung verschiedener kryptographischer Aufgaben.

3.6 Verdichtung zur BASIS VARIATION

Um die Bestimmungsbasis auf dezimale Größen zurückzuführen wird eine **Kontraktionsfunktion** eingeführt. Für die Ziffern der **Hashfunktionsfolge** wird das Zahlensystem zur **Basis 78** (Expansionsbasis +1) unterstellt.

Jeweils drei Ziffern der Hashfunktionsfolge werden seriell durch **MODULO 256** in dezimale Zahlen 0 bis 255 (ohne Wiederholung) zurückgerechnet. Der Parameter **Theta** wird abgezogen. Die Ergebnisse werden in der BASIS VARIATION gespeichert, einem Array von 16x16 Elementen. Eine rückwärts gerichtete Bestimmung vorhergehender Daten ist nicht möglich (zweite Einweg-Funktion).

Die ersten vier Rückrechnungen ab **Variante = 11** zeigen sich wie folgt:

3 Ziffern Basis 78	dezimal	Modulo 256	- Theta	Element
âë3	413559	119	3	116
ë3@	462682	90	3	87
3@j	23289	249	3	246
@jQ	392912	208	3	205

BASIS-VARIATION (256 Elemente)
Verteilung der Elemente

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

3.7 Berechnung der „CypherMatrix“

Für die Berechnung der CypherMatrix werden die Elemente der BASIS VARIATION in ihrer Verteilung 16x16 Zeichen direkt als Bestimmungsbasis verwendet. Dabei werden die Werte direkt auf die Indexwerte der Bytes von **0** bis **255** (vergleichbar: ASCII-Zeichensatz) bezogen.
Für die Verteilung der Zeichen (16x16) in der CypherMatrix bietet das Programm zwei Varianten: Variante **B** und Variante **D:**

3.7.1 Versetzter Austausch der Indizes: i,j (Variante B)

Dreifache Permutation der Elemente zur Berechnung der endgültigen Verteilung in der CypherMatrix:

Auszug aus dem Quellcode:

```
 N = Alpha - 1
 FOR s = 1 TO 3                                          drei Schleifen
    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)      Variation$(N)
              CASE 2
                 Matrix$(2,a,j)  = Matrix$(1,i,j)     Vertauschung
              CASE 3
                 Matrix$(3,i,a)  = Matrix$(2,i,j) endgültige Matrix
           END SELECT
        NEXT j
     NEXT i
  NEXT s
```

3.7.2 Dynamische Generierung der Indizes (Variante D)

Alle Indexwerte (16x16) werden in einem gesonderten Array **IndexFolge(2,16)** aus den Elementen der BASIS-VARIATION neu generiert (Anwendung der CypherMatrix Funktion):
Auszug aus dem Quellcode:

```
 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              Daten aus der BASIS-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)          Generierung der CypherMatrix
      INCR N
      IF N > 256 THEN N = 1
    NEXT J
 NEXT I
```

Endgültige CypherMatrix (Variation: D)

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

3.8 Steuerungsparameter

Ab Position Alpha = 204 werden 128 Zeichen als Chiffre-Alphabet des Bitsystems zur Basis 7 entnommen. Bestimmte Zeichen (Hex: 00 bis 20, 22, 2C, B0, B1, B2, D5, DB, DC, DD, DE, DF und weitere) werden ausgeklammert, weil sie in einigen Situationen noch ihre ursprünglichen Aufgaben wahrnehmen (zB. **1A** =ASCII-26) und die ordnungsgemäße Durchführung des Programms stören.

Chiffre-Alphabet (Basis 7)

1	ì ¾ | B º € Ó ¨ a “ Ê ú … Ä ¤ >	*16*
17	ô @ s / õ — › c d Œ 8 *ƒ* ü x •	*32*
33	g h i 6 ¦ z 7 [F „ µ – Ø ž	*48*
49	· œ † G ¡ © » ª % « 9 ¬ A : ¯ &	*64*
65	; ¼ < D H ¿ U ï I À Á J ã Ù æ O	*80*
81	P Â Ã Å Æ Ë Ö × é ê ë t W ö Í]	*96*
97	0 à C j á N 2 ~ Ô ` Q þ ÷ î k Î	*112*
113	X I = ð * ¸ ” Ÿ E u ¥ É ¢	*128*

Chiffre-Alphabet (hex)

1	EC	BE	7C	42	BA	80	D3	A8	61	93	CA	FA	85	C4	A4	3E	*16*
17	F4	40	73	2F	F5	97	9B	63	64	8C	81	38	83	FC	78	95	*32*
33	67	68	69	9D	36	A0	A6	7A	37	5B	46	84	B5	96	D8	9E	*48*
49	B7	9C	86	47	A1	A9	BB	AA	25	AB	39	AC	41	3A	AF	26	*64*
65	3B	BC	3C	44	48	BF	55	EF	49	C0	C1	4A	E3	D9	E6	4F	*80*
81	50	C2	C3	C5	C6	CB	D6	D7	E9	EA	EB	74	57	F6	CD	5D	*96*
97	30	E0	43	6A	E1	4E	32	7E	D4	60	51	FE	F7	EE	6B	CE	*112*
113	58	6C	3D	F0	2A	AD	B8	8D	94	9F	FF	45	75	A5	C9	A2	*128*

Block-Schlüssel

Der ab Position **149** der CypherMatrix entnommene Block-Schlüssel umfasst 42 Zeichen:

Z#M4?5¶û#,'Òå#ÏÐoq#ç‡è#pÑLK#}¹-Vù.í!³±fò§™

```
5A 0F 4D 34 3F 35 B6 FB 12 82 92 D2 E5 11 CF D0 6F 71 08 E7 87
E8 13 70 D1 4C 4B 18 7D B9 2D 56 F9 2E ED 21 B3 B1 66 F2 A7 99
```

Matrix-Schlüssel

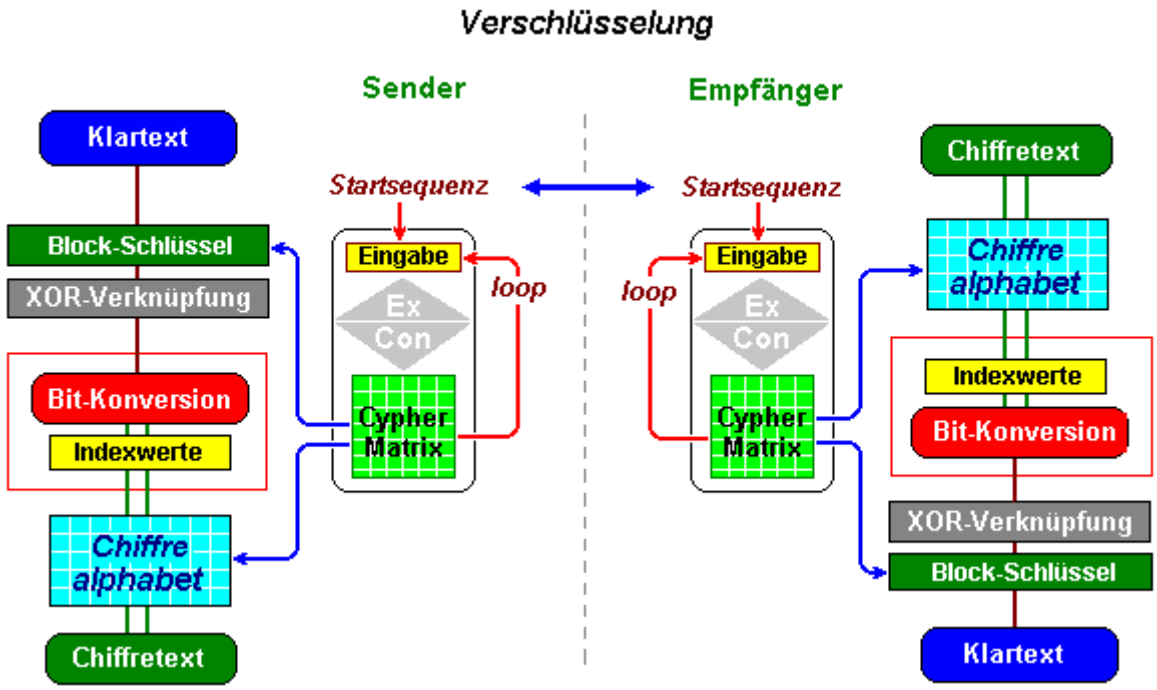
Als Startsequenz für die nächste Runde entnimmt das Verfahren ab Position **141** einen neuen Matrix-Schlüssel mit 42 Zeichen:

yÌŽ^T\®ñZ#M4?5¶û#,'Òå#ÏÐoq#ç‡è#pÑLK#}¹-Vù.

Der Matrix-Schlüssel wird auf den Beginn der Funktion zurückgeführt (**loop**). Die jeweiligen Matrix-Schlüssel steuern den gesamten Ablauf des Verfahrens, inhaltsgleich, sowohl beim Sender als auch beim Empfänger.

4 Der Codierbereich

Die Verschlüsselung – das Schreiben und Lesen von geheimen Informationen – findet ausschließlich im Codierbereich statt. Mit Eingabe der gleichen Startsequenz, sowohl beim Sender als auch beim Empfänger, werden im gesamten Verfahren ein identischer Verlauf und identische Steuerungsparameter erzeugt. Das folgende Schema zeigt die Zusammenhänge:



Die Verschlüsselung wird in folgenden Alternativen durchgeführt, und zwar:

1. **Basis-Coding:** Bit-Konversion allein ohne weitere Operationen oder
2. **Verbund-Coding:** Bit-Konversion mit zusätzlichen Operationen ,
   a) mit XOR-Verknüpfung (voran- oder nachgestellt),
   b) verbunden mit weiteren Operationen (vgl. "Combinierte Operationen")

Die Verschlüsselung ist im CypherMatrix Verfahren sehr einfach:

1. Der Generator erzeugt das erforderliche **System-Alphabet** und
2. mit der **Bit-Konversion** wird der Chiffretext geschrieben.

4.1 Programmübersicht

Vom Autor sind bisher die folgenden Verschlüsselungsprogramme nach dem CypherMatrix Verfahren entwickelt worden:

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 ParaCode	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	Crypto11	Coding11	ElvaCode	CM11Code StepCM11	1:1,46 1:1,46
12	4096	Crypto12	Coding12	MegaCode	MaxiCode	1:1,33

[1)] Programm mit drei Operationen (XOR – bit conversion - exchange),
[2)] Programm mit vier Operationen (dyn24 – XOR – bit conversion – exchange).

Außerdem sind noch weitere Verschlüsselungen im Bitsystem zur Basis 8 und in den Bitsystemen von zur Basis 9 bis zur Basis 14 wie folgt entwickelt worden:

System Basis	System Alphabet	Basis-Coding (ohne XOR)	Verbund-Coding (mit 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

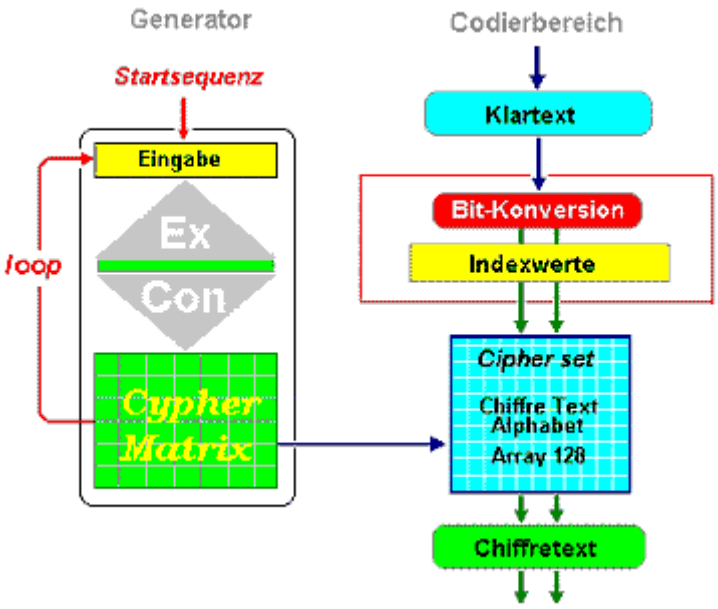
Das System-Alphabet für die Programme im Bitsystem zur Basis 8 umfasst 256 Zeichen.. Da für Bitsysteme ab Basis 9 und höher außer Unicode keine weiteren Einzelzeichen mehr zur Verfügung stehen, wurden für das jeweilige System-Alphabet die Ziffern des **Zahlensystems** zur **Basis 128** – das sind 16384 Ziffern – als **Doppelzeichen** verwendet.

Die Programme sind nach ihrer Bitsystematik (Basis 1 bis Basis 14) geordnet. Alle Programme sind Dos-Programme und laufen nur unter WindowsXP. Sie müssen auf **C#** umgeschrieben werden, wie dies bereits unter Leitung von Prof. **Bernhard Esslinger** (Uni Siegen) und seinem CrypTool-Team (insbesondere.: **Michael Schäfer**) mit den Programmen **DataCryp** und **DynaCode** geschehen ist [#8]. Nach dem Umschreiben laufen die C#-Programme 25mal schneller als die ursprünglichen DOS-Programme. Alle weiteren Programme müssen noch umgeschrieben werden.

4.2 „Basis-Coding“

Im „Basis-Coding“ erfolgt die Bit-Konversion direkt z.B. vom Bitsystem zur **Basis 8** des Klartextes zum Bitsystem zur **Basis 7** des Chiffretextes (alternativ: zur Basis 1 bis zur Basis 12). Aus der Bitfolge des eingelesenen Klartext-Blocks von 42 x 8 Bits (336 Bits) entsteht eine Bitfolge von 48 x 7 Bits (336 Bits). Die Units im Bitsystem zur Basis 7 stellen die Indices für die Chiffrezeichen im zugehörigen System-Alphabet dar. Es ist die einfachste Art digitaler Verschlüsselung. Zusätzlich ist nur der Generator erforderlich.

Die Indices holen die Chiffre-Zeichen aus dem jeweiligen System-Alphabet, die dann zum Chiffre-Text verbunden werden. Schematisch zeigen sich die Zusdammenhänge wie folgt:



Die Umwandlung vom Klartext zum codierten Text geschieht mit zwei Funktionen:

1. Bit-Konversion
   8-bit Klartextwerte → 7 bit Indexwerte (0 ...127)
2. Bestimmung des Chiffretexts
   7-bit Indexwerte → Chiffre-Alphabet (0...127) → Chiffretext.

Die Länge der Klartext-Blocks und Block-Schlüssel sind im vorliegenden Fall mit 42 Zeichen festgelegt. Die im Abschnitt 3.2 eingeführte Startsequenz bleibt unverändert:

Der schwarze Kater fängt immer graue Mäuse (42 Bytes)

Beispiel für die Verschlüsselung ist eine Ballade von Goethe: Datei **„Erlkonig.txt“** (633 Bytes).

Wer reitet so spät durch Nacht und Wind?
Es ist der Vater mit seinem Kind;
Er hat den Knaben wohl in dem Arm,
Er fasst ihn sicher, er hält ihn warm.

Mein Sohn, was birgst du so bang dein Gesicht?
Siehst, Vater, du den Erlkönig nicht?
Den Erlenkönig mit Krone und Schweif?
Mein Sohn, es ist ein Nebelstreif.

Du liebes Kind, komm, geh mit mir!
Gar schöne Spiele spiel ich mit dir;
Manch bunte Blumen sind an dem Strand,
Meine Mutter hat manch gülden Gewand.

Mein Vater, mein Vater, und hörest du nicht,
Was Erlenkönig mir leise verspricht?
Sei ruhig, bleibe ruhig, mein Kind;
In dürren Blättern säuselt der Wind.

Johann Wolfgang von Goethe 1782

Als ersten Block des zu verschlüsselnden Klartextes werden 42 Bytes eingelesen:

Wer reitet so spät durch Nacht und Wind?♪◘

```
57 65 72 20 72 65 69 74 65 74 20 73 6F 20 73 70 84 74 20 64 75
72 63 68 20 4E 61 63 68 74 20 75 6E 64 20 57 69 6E 64 3F 0D 0A
```

Der Klartext im Bitsystem zur **Basis 8** wird in Abschnitte des Bitsystems zur **Basis 7** umgewandelt. Die dezimalen Werte der umgewandelten Zeichen (Basis 7) sind Indexwerte für die Positionen der Zeichen im Chiffre-Alphabet. Die Indizes für das Chiffre-Alphabet müssen um +1 erhöht werden, da der Index „**0**“ im Array des Chiffre-Alphabets nicht erkannt wird.

Klartext

	W	e	r		r	e	i	t	
Basis 8:	**01010111**	**01100101**	**01110010**	**00100000**	**01110010**	**01100101**	**01101001**	**01110100**	**. . . . .**
Basis 7:	**0101011**	**1011001**	**0101110**	**0100010**	**0000011**	**1001001**	**1001010**	**1101001**	**0111010** **0** **. . . . .**
Index:	43	89	46	34	3	73	74	105	58 . . . .
(+1)	44	90	47	35	4	74	75	106	59 . . . .
Chiffretext:	„	ê	Ø	i	B	À	Á	`	9 . . . .

„êØiBÀÁ`9ŒØ<BÙÍgîÐ™Qb³  9îþ“UUîOkÖ¨W_<èŠ6<Ô¾¹8é¦$!¾Sù…¹3« . . . . . .

Hexadezimal

```
84 EA D8 69 42 C0 C1 60 39 8C D8 3C 42 D9 CD 67 EE D0 99 51 62
B3 8F 39 EE FE 93 55 55 EE 4F 6B D6 A8 57 5F 3C E8 8A 36 3C D4
BE B9 38 E9 A6 24 21 BE 53 F9 85 B9 33 AB . . . . . .
```

Der Chiffretext **Erlkonig.ctx** umfasst 736 Zeichen des System-Alphabets zur Basis 7.

4.3 „Verbund-Coding“

Beim Verbund-Coding werden verschiedene Operationen seriell verbunden, z.B. XOR-Operation mit Bit-Konversion und weiteren Operationen (dyn24, exchange).

4.3.1 XOR-Verknüpfung

Mit vorgeschalteter XOR-Verknüpfung vollzieht sich die Verschlüsselung in drei Funktionen:

1. Partielles dynamisches „One-time-pad“
   Klartext-Block → Block-Schlüssel → 8-bit XOR-Verknüpfung
2. Bit-Konversion
   8-bit XOR-Verknüpfung → 7 bit Indexwerte (0 ...127)
3. Bestimmung des Chiffretexts
   7-bit Indexwerte → Chiffre-Alphabet (0...127) → Chiffretext.

Im Quellcode werden die Module nacheinander durchlaufen:

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

4.3.2 Dynamisches „one-time-pad“

Als Beispiel werden die Verschlüsselungsschritte im Programm **DataCryp.exe** für die Datei **„Erlkonig.txt“** erläutert. In jeder Runde wird ein Klartext-Block von 42 Bytes (42x8 = 336 Bits) mit einem Block-Schlüssel von gleicher Länge XOR-Funktion verknüpft. Als ersten Klartext-Block liest das Programm folgende 42 Bytes ein:

Wer reitet so spät durch Nacht und Wind?♪◘

```
57 65 72 20 72 65 69 74 65 74 20 73 6F 20 73 70 84 74 20 64 75
72 63 68 20 4E 61 63 68 74 20 75 6E 64 20 57 69 6E 64 3F 0D 0A
```

Der ab Position **149** der CypherMatrix entnommene Block-Schlüssel umfasst:

Z#M4?5¶û#,'Òå#ÏÐoq#ç‡è#pÑLK#}¹-Vù.í!³±fò§™

```
5A 0F 4D 34 3F 35 B6 FB 12 82 92 D2 E5 11 CF D0 6F 71 08 E7 87
E8 13 70 D1 4C 4B 18 7D B9 2D 56 F9 2E ED 21 B3 B1 66 F2 A7 99
```

XOR-Verknüpfung:
Klartext:
01010111 01100101 01110010 00100000 01110010 01100101 01101001 01110100 . . . . .
Schlüssel:
01011010 00001111 01001101 00110100 00111111 00110101 10110110 11111011 . . . . .
XOR:
00001101 01101010 00111111 00010100 01001101 01010000 11011111 10001111 . . . . . .

Hex:	0D	6A	3F	14	4D	50	DF	8F . . . . .
dez:	13	106	63	20	77	80	223	143 . . . . .
ASCII:	□	j	?	#	M	P	ß	□ . . . . .

□j?#MPß□wö²¡Š1¼ ë#(ƒòšp#ñ#*{#Í #—JÍvÚß#Íª“

```
0D 6A 3F 14 4D 50 DF 8F 77 F6 B2 A1 8A 31 BC A0 EB 05 28 83 F2
9A 70 18 F1 02 2A 7B 15 CD 0D 23 97 4A CD 76 DA DF 02 CD AA 93
```

Als Ergebnis entsteht ein partielles **„one-time-pad“**. Klartext und Schlüssel sind gleich lang und der Schlüssel wird auch nicht wiederholt. In jeder Runde wird ein anderer Schlüssel aus der betreffenden CypherMatrix entnommen.

4.3.3 Bit-Konversion

Das Ergebnis der XOR-Verknüpfung im Bitsystem zur Basis 8 (42x8 = 336 Bits) wird in Zeichen des Bitsystems zur Basis 7 (336 Bits = 48x7) umgewandelt. Die dezimalen Werte der umgewandelten Zeichen (Basis 7) sind Indexwerte für die Positionen der Zeichen im Chiffre-Alphabet. Die Indizes für das Chiffre-Alphabet müssen um +1 erhöht werden, da der Index „**0**“ im Array des Chiffre-Alphabets nicht erkannt wird.

Bit-Konversion:
XOR Basis 8:
00001101 01101010 00111111 00010100 01001101 01010000 11011111 10001111 . . . . . .
Basis 7:
0000110 1011010 1000111 1110001 0100010 0110101 0100001 1011111 1000111 1 . . . . . .

Index:	6	90	71	113	34	53	33	95	71 . . . . .
(+1)	7	91	72	114	35	54	34	96	72 . . . . .
Alphabet (Basis 7)									
	Ó	ë	ï	l	i	©	h	]	ï . . . . .

Als Ergebnis der Bit-Konversion entsteht der Chiffretext der Datei **Erlkonig.ctx** (768 Zeichen)

> **Óëïli©h]ï**öÉþ—ÓõœÍ7ü·[ï¨=Ù*ƒ*B>aaÆEÊðhÃ*ƒ*ö—Ù¬»tX››Ë/
> MŸî@5PWí¸“QÐ?yš<ñ3o**□**øÉy¶€8Ð¦ÏšLÇÖ**□**z¾Qmc^¶^…i/U>
> ”;PI{aÏ9‡—ž'MxH3Ž>„–W<æÁL3f»¦–LèKÙ**□**ýï^¦Ã<6k¦DÔÖ
> 0æ2·„¿0³Œ!¨PÃ*ƒ*!4·j—õAË*ƒ*ÃJj'Œ×·Âåô·þ¾$7ç—¹3lÙ]Éï½
> . . . . . .

```
D3 EB EF 6C 69 A9 68 5D EF F6 C9 FE 97 D3 F5 9C CD 37 FC B7 5B 69
A8 3D D9 83 42 3E 61 61 C6 45 CA F0 68 C3 83 F6 97 D9 AC BB 74 58
9B BB CB 2F 0D 0A 4D 9F EE 40 35 50 57 AD ED B8 93 51 D0 3F 79 9A
3C F1 33 6F 90 F8 C9 79 B6 80 38 D0 A6 CF 9A 4C C7 D6 90 7A BE 51
6D 63 88 B6 88 85 69 2F . . . . . .
```

4.4 Dreifach Verbund

Die serielle Verknüpfung geschieht im Quellcode des Programms **CodeData.exe** wie folgt:

```
. . . . . .
CALL  XORGenerator (InputData$,TransData$)        für „XOR“
CALL  BitConversion (TransData$,DataTrans$)       für „Bit-Konversion“
CALL  ByteWechsel (DataTrans$,OutputData$)        für „exchange“
. . . . . .
```

4.5 Vierfache Verknüpfung

Im Quellcode des Programms **QuadCode.exe** besteht die folgende Schaltung:

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

In der gleichen Weise können weitere oder andere Operationen mit dem Programmlauf verbunden werden.

Eine Analyse weiterer Programme aus der Übersicht in Abschnitt 4.1 ist als Ergänzung zum Paradigma in der Kryptographie diesem Artikel als Anhang beigefügt. Um das Verfahren einmal zu testen, können Sie sich das Programm **DataCryp** mit einer verschlüsselten Datei **Message8.ctx** als **Datatest.zip** auf Ihren Rechner holen und versuchen, die Information zu entziffern.

5 Entschlüsselung

Für die Entschlüsselung erzeugt der Generator einen inhaltsgleichen Ablauf, wie bei der Verschlüsselung. Die Entschlüsselung wird im Codierbereich abgearbeitet, nur in der umgekehrten Reihenfolge:

1. Analyse des Chiffretextes
   Chiffretext → Chiffre-Alphabet (0...127) → 7 bit Indexwerte
2. Bit-Konversion
   7 bit Indexwerte (0 ...127) → 8-bit XOR-Verknüpfung
3. XOR-Verknüpfung
   8-bit XOR-Verknüpfung → Block-Schlüssel → Klartext-Block

Aus Blöcken von **48** Zeichen Chiffretext sucht das Verfahren im identisch erzeugten Chiffre-Alphabet die dezimalen Index-Werte der einzelnen Zeichen und verbindet deren binäre Zahlen zu einer Bitfolge von 336 Bits. Diese Bitfolge wird wiederum in **42** 8-bit Sequenzen (336 Bits) im Bitsystem zur Basis 8 aufgeteilt und mit dem entsprechenden Block-Schlüssel XOR-verknüpft. Als Ergebnis erscheint der ursprüngliche Klartext.

6 Sicherheit des Verfahrens

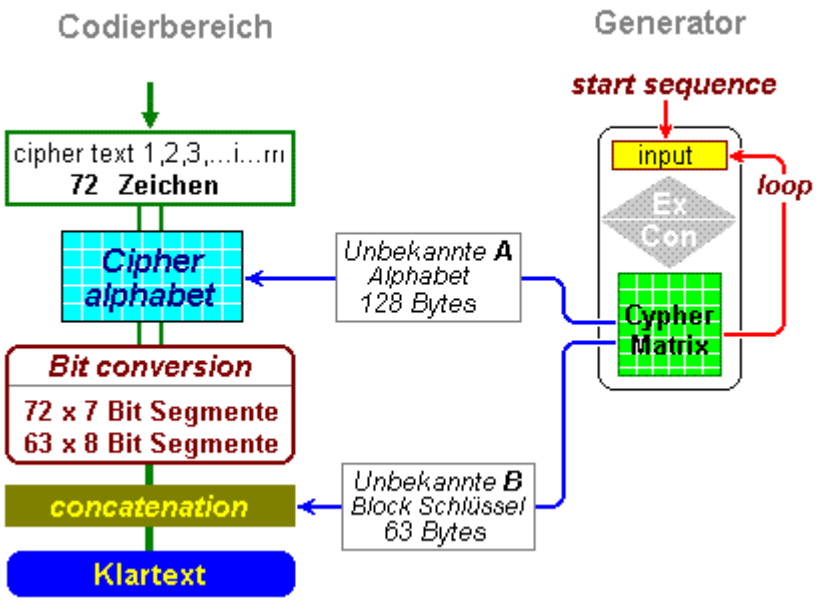
Zu den bekanntesten Angriffen gehören Strukturanalyse, „known plaintext attack“ und „chosen plaintext attack“, eventuell auch noch „differenzielle“ und „lineare“ Kryptoanalyse. Mit diesen Angriffen sollen aus dem Chiffretext statistisch erfassbare Regelmäßigkeiten herausgefiltert werden, die möglicherweise einen Weg zum Klartext aufzeigen. Zu den Auffälligkeiten der Sprache zählen Wiederholungsmuster und Wortkombinationen, Häufigkeitsstrukturen und Bigramme [#9].

Eine Analyse aller dieser Merkmale setzt allerdings voraus, dass Klartext und Chiffretext sich auch vergleichen lassen. Insoweit muss ein einheitliches Ordnungssystem bestehen, das sowohl im Klartext als auch im Chiffretext wirksam ist. Bei fast allen herkömmlichen Verfahren haben Klartext und Chiffretext die gleiche Länge: **"Längenkongruenz"** [#9]. Daraus folgt, dass für jedes Klartextzeichen auch ein bestimmtes Chiffrezeichen vorhanden sein muss. Damit arbeiten beide Bereiche im gleichen Bitsystem zur Basis 8, so dass ein einheitliches Ordnungssystem besteht und die beiden Bereiche sich auch vergleichen lassen.

Bei Verschlüsselungen nach dem CypherMatrix Verfahren dagegen findet ein Wechsel im Bitsystem statt. Im Chiffretext ensteht ein neues Ordnungssystem mit der Folge, dass beide Bereiche sich nicht mehr vergleichen lassen. Es fehlt die Längenkongruenz. Durch die Umwandlung der Zeichen vom Bitsystem zur Basis 8 in Zeichen zur Basis 7 entfällt auf jedes Klartextzeichen ein Chiffrezeichen, das um den Faktor 1,143 (8/7) länger ist als das zugehörige Klartextzeichen. Da beide Bereiche sich nicht vergleichen lassen, fehlt die Basis für alle herkömmlichen Angriffszenarien. Sie sind **wirkungslos** und wir können sie vergessen [#10].

Immerhin bleibt noch die Möglichkeit einer „brut force“ Attacke. Einem Angreifer sind grundsätzlich nur der **Chiffretext** und das **CypherMatrix** Verfahren bekannt. Das jeweilige Programm und die einzelnen Steuerungsparameter, einschließlich der Startsequenz, kennt er nicht. Er könnte also nur eine Iteration aller Möglichkeiten versuchen. Bei einem Angriff auf die Startsequenz mit 42 Bytes Länge ergibt sich eine Entropie von **336** und eine exponentielle Komplexität von **O(2^336)** = 1.4E+101. Ein Angreifer kann dann versuchen vom Chiffretext ausgehend, retrograd durch Iteration die einzelnen Stufen der Entschlüsselung herauszufinden.

Die Entschlüsselung geschieht in jedem Durchgang wie folgt:



Das Verfahren enthält drei Funktionen:

1. Klartextblock --> **Block-Schlüssel** --> -8 bit XOR-Sequenzen
2. 8-bit XOR Sequenzen --> 7-bit Index-Werte
3. 7-bit Index-Werte --> **Chiffre-Alphabet (128)** --> Chiffretext

In diesen Funktionen sind die Parameter **Block-Schlüssel** und **Chiffre-Alphabet** zwei voneinander unabhängige Variable. Es gelten:

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

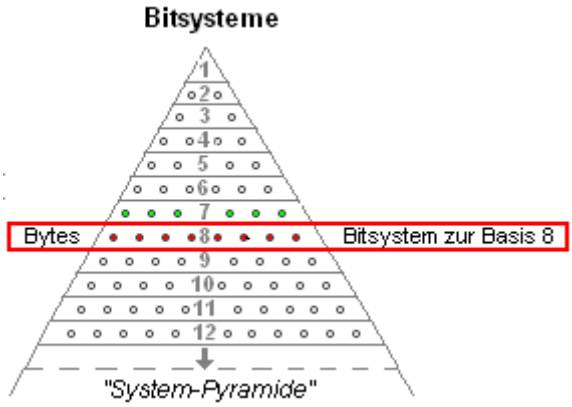
fx = funktionale Verbindung
an = Klartext
k1 = **Block-Schlüssel**
b1 = 8-bit Sequenz
b2 = 7-bit Index-Wert
k2 = **Chiffre-Alphabet (128)**
cm = Chiffretext

Die Ermittlung des Chiffretextes **cm** und die retrograde Suche nach dem Klartext **an** zeigen sich somit als Gleichungen mit zwei unbekannten Veränderlichen: **k1** und **k2**. Das führt bekanntlich nur dann zu einer eindeutigen Lösung, wenn eine Unbekannte aus der anderen abgeleitet werden kann oder wenn zwei Gleichungen mit denselben Unbekannten vorhanden sind.

Aber zwischen dem jeweiligen Block-Schlüssel = k1 und dem in derselben Runde generierten Chiffre-Alphabet (128) = k2 gibt es keine Verbindung. Beide sind zwar aus der aktuellen CypherMatrix entnommen, haben aber keine funktionale Beziehung: (k1 --> (**Hk MOD 169**)+1) und k2 --> (**Hk +Hp**) **MOD 255**)+1). Die Runden CypherMatrix selbst ist aus der ursprünglichen Start-Sequenz hergeleitet. Dahin führt jedoch kein Weg zurück (zwei Einwegfunktionen stehen dagegen). Es gibt somit viele Paare **Chiffre-Alphabet** / **Block-Schlüssel**, die nach einem versuchten "brute force" Angriff irgendwelche lesbaren Texte liefern, von denen man aber nicht weiß, welcher der Richtige ist: Angriff mit "brute force" . Somit kann auch „brute force“ keinen Erfolg haben.

7 Zusammenfassung

Im Bereich der dynamischen Bitfolgen ergeben sich folgende Zusammenhänge und Strukturen: Bitfolgen müssen in definierte Abschnitte (Längen) geteilt und systematisiert werden. Als Basis bietet sich die Länge an: **Bitsystem** zur **Basis** der definierten **Länge**.



Zur Visualisierung der Abläufe in einem bestimmten Bitsystem muss ein unabhängiges System-Alphabet definiert werden. Für Basis 1 entspricht das System-Alphabet dem Binaersystem mit 2 Bits. Zu Beginn der Computertechnik ist das einfache ASCII-System mit 128 Zeichen festgelegt worden. Als sich das als zu eng abzeichnete, wurde das System-Alphabet für Basis 8 auf das erweiterte ASCII-System mit 256 Zeichen ausgedehnt. Für das Bitsystem zur Basis 16 besteht das System-Alphabet: **Unicode**.

Heute wird fast immer im Bitsystem zur Basis 8 gearbeitet, so auch bei der Verschlüsselung. Der Klartext im Bitsystem zur Basis 8 wird zum Chiffretext verschlüsselt, der ebenfalls im Bitsystem zur Basis 8 ausgegeben wird. Dies erhellt schon aus der Tatsache, dass der Chiffretext die gleiche Länge hat – oder haben soll – wie der Klartext [#12]. Also in beiden Fällen **8 Bits**. Ein Systemwechsel findet dementsprechend nicht statt.

Im CypherMatrix Verfahren dagegen geschieht die Verschlüsselung immer durch einen Wechsel vom Bitsystem zur Basis 8 in ein anderes Bitsystem, am besten in das Bitsystem zur Basis 7. Der Chiffretext umfasst hier die **1,143-**fache Länge des Klartextes. Dabei ist die Definition des System-Alphabets und die Steuerung der Verschlüsselung allein Aufgabe des **Generators**. Das System-Alphabet allein bestimmt den Chiffretext. Ein **Schlüssel** im herkömmlichen Sinne zur Einbindung in die Verschlüsselung ist nicht erforderlich. Nur zur Initiierung des Generators wird eine **Startsequenz** (Passphrase) von optimal 42 Bytes eingesetzt. Insoweit scheint es sinnvoller zu sein, von „Kodierung“ zu sprechen anstatt von „Verschlüsselung“.

Weitere Einzelheiten zum **CypherMatrix** Verfahren unter: www.telecypher.net/
Wer sich intensiver mit dem Verfahren beschäftigen möchte, kann einzelne Programme mit oder ohne Source-Code beim Autor per e-mail anfordern und unter der CMLizenz damit arbeiten (eschnoor@multi-matrix.de).

München, im August 2012

Hinweise

[#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) Der kern des CypherMatrix Verfahrens, www.telecypher.net/CYPHKERN.HTM