(19) Republik Österreich Patentamt

(11) Nummer: AT 004 041 U1

(12) GEBRAUCHSMUSTERSCHRIFT

(21) Anmeldenummer: 715/99

(22) Anmeldetag: 15.10.1999

(42) Beginn der Schutzdauer: 15.11.2000

(45) Ausgabetag: 27.12.2000

(51) Int.Cl.[7] : G06F 7/58
H04L 9/20, 9/18

(73) Gebrauchsmusterinhaber:

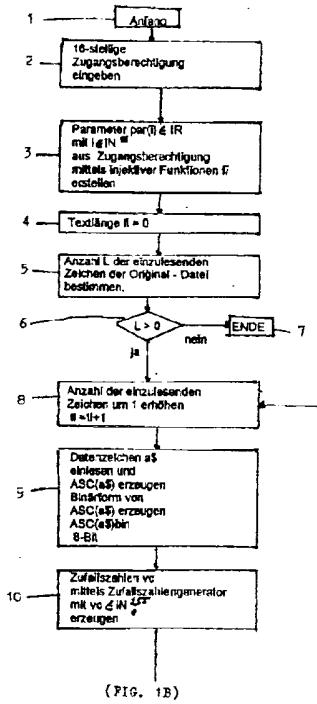
WAIS JULIUS
A-3860 HEIDENREICHSTEIN, NIEDERÖSTERREICH (AT).

(72) Erfinder:

WAIS JULIUS
HEIDENREICHSTEIN, NIEDERÖSTERREICH (AT).

(54) VERFAHREN UND EINRICHTUNG ZUM VERSCHLÜSSELN BZW. ENTSCHLÜSSELN VON DATEN

(57) Zum Verschlüsseln von Daten, die aus einer Anzahl von Zeichen bestehen, wird für jedes Zeichen eine Zufallszahl generiert, aus der mittels einer vorgegebenen, injektiven Funktion eine Codezahl erzeugt wird, mit der das jeweilige Zeichen über eine EXKLUSIV-ODER-Funktion zu einem verschlüsselten Zeichen logisch verknüpft wird. Die verschlüsselten Zeichen werden zusammen mit den Zufallszahlen in einem Feld gespeichert, das mit komplementären Bits aufgefüllt wird. Zum Entschlüsseln werden dann aus den zusammen mit den verschlüsselten Zeichen übermittelten Zufallszahlen mittels der selben vorgegebenen injektiven Funktion die jeweiligen Codezahlen erzeugt; die dann mit den zugehörigen verschlüsselten Zeichen über die EXKLUSIV-ODER-Funktion logisch verknüpft werden, um die ursprünglichen Zeichen zu erhalten.



(FIG. 1B)

(FIG. 1B)

AT 004 041 U1

Die Erfindung betrifft ein Verfahren zum Verschlüsseln von Daten, die aus einer Anzahl von Zeichen bestehen, sowie ein Verfahren zum Entschlüsseln von mit einem solchen Verfahren verschlüsselten Daten.

Weiters bezieht sich die Erfindung auf eine entsprechende Einrichtung zum Ver- bzw. Entschlüsseln von Daten sowie auch auf einem Programmdatenträger, wie eine Diskette oder eine CD-ROM, mit einem Programm zur Durchführung einer derartigen Ver- und Entschlüsselung von Daten.

Es gibt bereits die verschiedensten Techniken zum Verschlüsseln von Daten, die aber in der Regel auf einer aufwendigen Verschlüsselungs- und Entschlüsselungs-Software beruhen, die oft verhältnismäßig lange Rechnerzeiten benötigt und auch nicht sicher ist.

Ziel der Erfindung ist es daher, eine Technik zum Verschlüsseln bzw. Entschlüsseln von Daten vorzusehen, die einfach ist, wenig Rechnerzeit erfordert und dabei nichtsdestoweniger sicher bezüglich eines unbefugten Zugriffs ist.

Das erfindungsgemäße Verschlüsselungs-Verfahren ist dadurch gekennzeichnet, dass für jedes Zeichen der zu verschlüsselnden Daten eine Zufallszahl generiert wird, aus der mittels einer vorgegebenen, injektiven Funktion eine Codezahl erzeugt wird, mit der das jeweilige Zeichen gemäß der EXKLUSIV-ODER-Funktion zu einem verschlüsselten Zeichen logisch verknüpft wird, und dass die verschlüsselten Zeichen zusammen mit den zugehörigen Zufallszahlen in binärer Form jeweils in einem Feld gespeichert bzw. übertragen werden, das eine Feldlänge aufweist, die doppelt so groß ist wie die Zahl der Bits der jeweiligen verschlüsselten Zeichen und dazugehörigen Zufallszahlen zusammen, wobei die restlichen Plätze des jeweiligen Feldes mit den zu den Bits der verschlüsselten Zeichen und der Zufallszahl komplementären Bits aufgefüllt werden.

Da die Codezahl bei der vorliegenden Verschlüsselungstechnik für jedes zu verschlüsselnde Zeichen aus einer eigenen Zufallszahl mittels einer eindeutig abbildenden Funktion hergeleitet wird, ist auch jede Codezahl zufällig. Jedes zu verschlüsselnde Zeichen wird mit einer solchen ihm eigens zugeordneten, zufälligen Codezahl logisch über die EXKLUSIV-ODER-Funktion (nachstehend kurz XOR-Funktion) verknüpft. Diese XOR-Funktion hat den Vorteil, dass sie in gleicher Weise auch bei der Rücktransformation der verschlüsselten

Zeichen zu den ursprünglichen, unverschlüsselten Zeichen mit Hilfe derselben Codezahl verwendet werden kann. Die injektive Funktion, mit der aus den Zufallszahlen die Codezahlen erzeugt werden, ist an sich ein beliebiger Algorithmus, es ist nur notwendig, dass jeder Zufallszahl eindeutig eine Codezahl zugeordnet wird. Im Hinblick auf eine spätere einfache Entschlüsselung der verschlüsselten Zeichen werden die der Verschlüsselung zu Grunde gelegten zufälligen Zufallszahlen zusammen mit den verschlüsselten Zeichen gespeichert bzw. einem Empfänger übermittelt. Um hierbei einem unbefugten Entschlüsselungsversuch entgegenzuwirken, werden die verschlüsselten Zeichen samt den zugehörigen Zufallszahlen „neutralisiert", und demgemäß sieht die Erfindung weiters vor, dass ein Feld vorgesehen wird, das durch komplementäre Bits aufgefüllt wird. Jedes gesamte Feld enthält daher gleich viele „1"-Bits und „0"-Bits. Dadurch ist es nicht möglich, aus der jeweiligen Anzahl aus „1"-Bits und „0"-Bits Rückschlüsse auf die benutzten Funktionen oder Algorithmen zu ziehen, da immer gleich viele „0"-Bits und „1"-Bits im Feld enthalten sind.

Zum „Verstecken" der Bits, d.h. zum Mischen der Stellen der Bits der verschlüsselten Zeichen und Zufallszahlen, ist es dabei im Hinblick auf eine zusätzlich erhöhte Sicherheit auch vorteilhaft, wenn die Plätze des Feldes für die Bits des jeweiligen verschlüsselten Zeichens und der jeweiligen Zufallszahl auf Basis einer vorgegebenen bijektiven Funktion, verschieden von der ursprünglichen Reihung der Bits, festgelegt werden. Im Fall von beispielsweise jeweils acht Bit langen verschlüsselten Zeichen und Zufallszahlen enthält daher jedes Feld beispielsweise 2 mal 16 Bits, wobei auf beliebigen Plätzen n, entsprechend der bijektiven Funktion „gemixt", die Bits der verschlüsselten Zeichen und Zufallszahlen gespeichert werden: Beispielsweise kann sich aufgrund dieser bijektiven Funktion ergeben, dass das erste Bit des verschlüsselten Zeichens an die siebente Stelle des Feldes kommt, das zweite Bit an die zehnte Stelle usw., wobei diese Umreihung (also die bijektive Funktion) von Feld zu Feld beibehalten werden kann. Auf den Stellen „n + 16" (modulo 32) des Feldes werden wie erwähnt die jeweils komplementären Bits gespeichert, d.h. wenn an einer Stelle „1" oder „10" des Feldes ein „1"-Bit gespeichert wird, wird komplementär dazu an der 17. oder 26. Stelle des Feldes ein „0"-Bit gespeichert usw..

Zur zusätzlichen Erhöhung der Sicherheit kann als bijektive Funktion eine von einem eingegebenen Benutzerkennwort abhängige Funktion vorgegeben sein. Dies ist vor allem dann zweckmäßig, wenn auf einem eigenen Rechner bzw. auf einer eigenen Diskette Dateien gespeichert werden, die gegen unbefugten Zugriff zu schützen sind. Selbstverständlich

können jedoch die verschlüsselten Daten auch übertragen werden, wobei im letzteren Fall der Empfänger das Benutzerkennwort kennen muß, um eine Entschlüsselung vornehmen zu können, wie noch erläutert werden wird.

In entsprechender Weise kann auch die vorgegebene injektive Funktion für die Erzeugung der Codezahlen von einem eingegebenen Benutzerkennwort abhängig festgelegt werden.

Ferner kann die vorgegebene injektive Funktion für die Erzeugung der Codezahlen als von der Anzahl der Zeichen der zu verschlüsselnden Daten abhängige Funktion festgelegt werden. Dadurch ändert sich der Algorithmus für die Erzeugung des Codezahlen aus den Zufallszahlen von Datensatz zu Datensatz, was ebenfalls zur Sicherheit der Verschüsselung beiträgt.

Die Zufallszahlen können in an sich bekannter Weise mit einem hardwaremäßig realisierten Zufallsgenerator, beispielsweise unter Auswertung des Weißen Rauschens einer Diode oder eines Widerstandes, oder aber mit einer bekannten Software erzeugt werden. Im Hinblick auf die bevorzugte Anwendung einer 8-Bit-Codierung sind die generierten Zufallszahlen vorzugsweise natürliche Zahlen zwischen 0 und 255.

Zum Entschlüsseln von wie vorstehend angegeben verschlüsselten Daten wird derart vorgegangen, dass aus den zusammen mit den verschlüsselten Zeichen übermittelten Zufallszahlen mittels derselben vorgegebenen injektiven Funktion die jeweiligen Codezahlen erzeugt werden, die dann mit den zugehörigen verschlüsselten Zeichen über die XOR-Funktion logisch verknüpft werden, um die ursprünglichen Zeichen zu erhalten.

Sofern die verschlüsselten Daten in einem mit komplementären Bits aufgefüllten Feld, wie vorstehend erläutert, derart übermittelt werden, dass die Reihung der Bits geändert wurde, wird erfindungsgemäß weiters vorgesehen, dass unter Anwendung der genannten bijektiven Funktion die in den jeweiligen Feldern in verschiedenen Reihungen enthaltenen Bits der jeweiligen verschlüsselten Zeichen und zugehörigen Zufallszahlen in der ursprünglichen Reihung ermittelt werden, bevor die Codezahlen erzeugt und mit den verschüsselten Zeichen logisch verknüpft werden.

Die Erfindung schafft somit eine neue, vorteilhafte Technik zum Verschlüsseln bzw. Entschlüsseln von Daten, wobei ein Verfahren bzw. eine Programmlogik in vergleichbarer Weise wie eine Einrichtung, also ein entsprechend eingerichteter Rechner, betroffen sind.

Die Erfindung wird nachfolgend anhand von bevorzugten Ausführungsbeispielen, auf die sie jedoch nicht beschränkt sein soll, und unter Bezugnahme auf die Zeichnung noch weiter erläutert. Es zeigen: Die Fig. 1 in den zusammengehörenden Teilfiguren 1A, 1B, 1C zusammen ein Ablaufdiagramm zur Veranschaulichung eines Verschlüsselungsvorganges; Fig.2 schematisch die Zuordnung der Bits der verschlüsselten Zeichen und der zugehörigen Zufallszahlen in ein Doppel-Feld, unter Auffüllung dieses Feldes mit komplementären Bits; und Fig. 3 in den zusammengehörenden Teilfiguren 3A, 3B ein Ablaufdiagramm für den Entschlüsselungsvorgang.

In der nachfolgenden Beschreibung werden Zeichen bzw. Zahlen im ASCII-Code mit „ASC..." bezeichnet; in binärer Form werden die Zeichen bzw. Zahlen mit dem Zusatz „...bin" angegeben; die alphanumerischen Zeichen der zu verschlüsselnden Daten werden mit „a$" bzw. „a1$", „a2$" bzw. bezeichnet, die Zufallszahlen (natürliche Zahlen zwischen 0 und 255) werden mit „vc" bezeichnet; aus diesen Zufallszahlen vc wird über eine vorgegebene injektive Funktion ALG1 jeweils eine zugehöre Zahl „code" erzeugt; mit diesen „code"-Zahlen werden die Datenzeichen „a$" logisch verknüpft, so dass die verschlüsselten Zeichen „vz" erhalten werden. Bevorzugt wird ein 8 Bit – Format für die jeweiligen Zeichen bzw. Zahlen verwendet.

Beim Verschlüsseln von Daten, vgl. Fig. 1, wird gemäß dem vorliegenden Ausführungsbeispiel derart vorgegangen, dass nach einem Startschritt 1 (Fig. 1A) bei 2 ein m-stelliges, z. B. 16-stelliges Benutzerkennwort PW (Zugangsberechtigung, „Password") eingegeben wird, das bei der nachfolgenden Verschlüsselung von Daten mit verwendet wird.

Aus diesem Benutzerkennwort PW, genauer aus dem ASCII-Code der Einzelzeichen hievon, werden mittels injektiver Funktionen fi reelle Zahlen par(i), mit $1 \leq i \leq m$, erzeugt. Ein Beispiel für die verwendete injektive Funktion ist die Sinus-Funktion, bei deren Anwendung der Wertebereich auf maximal ±1 beschränkt wird; wenn der Betrag der Funktion verwendet wird, liegen die Werte zwischen 0 und 1:

$$par(i) = ABS[SIN((i/100) + ASC(Pwi\$))]$$

Da ASC(PWi$) immer eine ganze Zahl im Intervall {0,255}und weiters das Vielfache der Zahl k . π (3,14159) keine ganze Zahl für $1 \leq k \leq m$ ist, wird damit eine Funktion definiert, welche im Wertebereich für jedes feste i injektiv ist.

Vor dem Einlesen des zu verschlüsselnden Textes oder allgemein der zu verschlüsselnden Datenzeichen a$ wird im Ablauf gemäß Fig. 1 noch in einem Schritt 4 die Textlänge tl gleich 0 definiert, und es wird dann bei 5 die Anzahl L der einzulesenden, d.h. zu verschlüsselnden Datenzeichen ermittelt. Bei 6 wird dann vorsorglich abgefragt, ob zu verschlüsselnde Datenzeichen überhaupt vorliegen (L > 0 ?), und wenn nein, wird bei 7 zum Programm-Ende gegangen. Bei L > 0 wird hingegen im nachfolgenden Schritt 8 die Textlänge tl inkrementiert (tl = tl + 1), und es werden dann gemäß Block 9 die Datenzeichen a$ eingelesen und aus ihrem ASCII-Format ASC(a$) Binärformen ASC(a$)bin erzeugt.

Wie erwähnt wird für jedes eingelesene Zeichen a$ durch einen Zufallszahlen-Generator eine natürliche Zahl vc, mit $0 \leq vc \leq 255$, erzeugt. Daher kann die Binärform vcbin aus 8 Bits bestehen. Dies ist in Fig. 1A bei 10 veranschaulicht; wobei gemäß Schritt 11 (s. Fig. 1B) die binäre Form der jeweiligen Zufallszahl – vcbin – erzeugt wird.

Über eine injektive Funktion ALG1, welche von der Anzahl L der eingelesenen Zeichen a$ sowie von par(i), mit $1 \leq i \leq m$, abhängig ist, wird eine zur Zufallszahl vc gehörende Zahl „code", mit $0 \leq code \leq 255$, erzeugt. Deshalb besteht die Binärform hievon, codebin, aus 8 Bits. Da „code" aus der Zufallszahl vc erzeugt wurde, ist „code" selbst vom Zufall abhängig. Dieser Vorgang ist in Fig. 1B bei 12 gezeigt.

Als Beispiel für die injektive Funktion ALG1 kann eine Exponentialfunktion verwendet werden. Vorweg wird die Zufallszahl vc durch 256 dividiert:

$$hz = vc/256$$

Die dadurch erhaltene reelle Zahl hz ist selbstverständlich ebenfalls zufällig, und sie ist ≤ 1.

Hieraus können Werte y' gemäß

$$y' = 2^{hz}$$

berechnet werden, wobei $y' \leq 2$ und ≥ 0 ist.

Um zusätzlich die Sicherheit zu erhöhen, kann ergänzend die Anzahl L der eingegebenen Datenzeichen mit verwendet werden, um Werte y zu berechnen, u.zw. gemäß der Beziehung

$$y = y' \cdot ABS[SIN(L)].$$

Da hier ABS[SIN(L)] zwischen 0 und 1 liegt, liegt auch der jeweilige Wert y (wie y') zwischen 0 und 2.

Mittels der Beziehung

$$x = y - INT(y)$$

werden nun reelle Zahlen x im Intervall [0,1] erzeugt, die zur Erzeugung der Zahlen „code" (aus den Zufallszahlen vc) verwendet werden:

$$code = INT(1000 \cdot x) \bmod 255$$

Somit wird eine ganze Zahl „code" erhalten, die aus der Zufallszahl vc hergeleitet und von der Anzahl L der eingegebenen Zeichen abhängig ist.

Gemäß Schritt 13 in Fig. 1B wird sodann die Binärform codebin der Zahl code erzeugt, und im Schritt 14 folgt dann die logische Verknüpfung zur Verschlüsselung des jeweiligen Zeichens a$, u.zw. gemäß einer EXKLUSIV-ODER-Funktion (XOR-Funktion):

$$vzbin = ASC(a\$)bin \text{ XOR } codebin.$$

Da die Zahl code (bzw. codebin) eine Zufallszahl ist, ist auch vzbin – ein 8 Bit-Zeichen – zufällig.

Gemäß Schritt 15 in Fig. 1B wird nun eine Permutation P von natürlichen Zahlen zwischen 1 und 32 erzeugt ($1 \leq n(i) \leq 32$), die als Basis für ein „Mixen" und „Verstecken" der 8 Bit-Wörter vzbin und vcbin (die zusammen 16 Bits aufweisen) in einem 32 Bit-Feld dient, vgl. auch die nachfolgenden Schritte 16 (in Fig. 1B) und 17 bis 21 (in Fig. 1C) sowie das Schema gemäß Fig. 2.

Im einzelnen werden dabei gemäß Block 16 (Fig. 1B) die 8 Bits vzbin als s(n(i)) an Stellen n(1) bis n(8) in einem 32 Bit-Feld s und gemäß Block 17 (Fig. 1C) die 8 Bits von vcbin bin als s(i) an Stellen n(9) bis n(16) des Feldes s gesetzt, wobei die Stellen n(i) entsprechend der Permutation P festgelegt werden; an die Stellen n(i + 16) des Feldes s werden die jeweils zu den Bits von vzbin und vcbin komplementären Bits gesetzt, vgl. die Schleife in Fig. 1C

mit den Feldern 18 bis 21, wobei bei 18 abgefragt wird, ob das jeweilige Bit für s(i), mit i = 1,2...16, ein „1"-Bit oder ein „0"-Bit ist; je nachdem wird das Komplement – Bit an der Stelle n(i + 16) als „0"-Bit (Feld 19) oder als „1"-Bit (Feld 20) festgelegt und gemäß Feld 21 im s-Feld gespeichert. Damit ergibt sich zwangsläufig, dass das Feld s immer 16 „1"-Bits und 16 „0"-Bits enthält, so dass Rückschlüsse auf das jeweilige verschlüsselte Zeichen bzw. auf die verwendete Funktion auf Grund einer Wahrscheinlichkeits-Rechnung (wie dies bei nicht gleich vielen „1"- und „0"-Bits möglich wäre) unmöglich sind.

In Fig. 2 ist dieser Vorgang des Mischens und Versteckens schematisch veranschaulicht, wobei gezeigt ist, dass die Bits von vzbin und vcbin in einer von der Permutation P abhängigen Weise an andere Stellen n(i) im Feld s umgereiht werden, wobei die jeweils komplementären Bits an die Stellen n(i + 16) im Feld s gesetzt werden.

Die Permutation P kann beispielsweise wie folgt abhängig von der Anzahl L der eingegebenen Zeichen a$ und von den m (=16) Zeichen des Benutzerkennworts PW bestimmt werden, wobei die wie oben beschrieben abgeleiteten reellen Zahlen par(i) zu Grunde gelegt werden können (mit $0 \leq i \leq 16$):
Es wird eine natürliche Zahl j wie folgt ermittelt:

$$j = 1 + L \bmod 16 \qquad \text{mit } 1 \leq j \leq 16.$$

Beschränkt sich die Permutation P auf die Plätze n(i) mit i 1,2,.....,31,32, so kann man, mit L = Anzahl der eingegebenen Zeichen, die Plätze n(i) der Permutation P beispielsweise in einer FOR-NEXT-Schleife abhängig vom Benutzerkennwort PW und der Anzahl der eingegebenen Zeichen L etwa wie bestimmen. Dabei wird eine Zahl zr berechnet:

$$zr = 100 \,.\, par(j) \text{ mit } 1 \leq j \leq 16$$

Hierbei ist zr eine reelle Zahl aus dem Intervall [0,100], da par(j) mit $1 \leq j \leq 16$ eine reelle Zahl aus dem Intervall [0,1] ist. Da par(j) aus dem Benutzerkennwort PW hergeleitet wurde, ist damit zr vom Benutzerkennwort abhängig.

Wenn nun m als Startwert der FOR-NEXT-Schleife definiert wird, so kann weiters beispielsweise

$$m = LOG(L + zr)$$

gesetzt werden, wobei rn eine reelle Zahl mit rn ≥ 0 ist (LOG ist eine streng monoton wachsende Funktion mit LOG(1) =0). Die Zahl rn ist daher vom Benutzerkennwort PW und der Anzahl L der eingegebenen Zeichen a$ abhängig. Mit

$$rn' = rn - INT(rn) \text{ bzw.}$$

$$r = 0{,}9883 \cdot rn' + 0{,}3$$

wird nun der Startwert für die Permutation erstellt ($0 \leq rn < 1$), die sich auf 32 Plätze beschränkt, was den maximalen Index festlegt. (Die Zahlen 0,9883 und 0,3 sind willkürlich gewählt.)

In den folgenden FOR-NEXT-Schleifen werden für einen jeweils um 1 reduzierten Index (ausgehend vom maximalen Index) Zahlen z mit

$$z = r - INT(r)$$

berechnet, wobei $0 < z < 1$ gilt. Hieraus wird die Platz-Zahl n gemäß der Beziehung

$$n = 1 + INT(z \cdot Index)$$

berechnet, wobei „Index" schrittweise (pro Schleifen – Durchlauf) von 32 auf 2 erniedrigt wird.

Die Bits des Feldes s – die insgesamt ein Zeichen a$ darstellen – werden anschließend, nach einer eventuellen Datenkompression, in einer „Codedatei", d.h. in einer Verschlüsselungsdatei, gespeichert (Festplatte bzw. Diskette etc.), s. Feld 22 in Fig. 1C. Im Anschluß daran wird bei 23 abgefragt, ab alle Datenzeichen verschlüsselt wurden (tl < L ?); wenn noch Datenzeichen übrig sind (tl < L), wird zu Schritt 8 in Fig. 1A zurückgekehrt; ansonsten wird die Verschlüsselung bei 24 beendet.

Beim Entschlüsseln, s. Fig.3 (Fig. 3A + 3B), wird auf analoge Weise vorgegangen, wobei nach einem Startschritt 31 wiederum – bei 32 – die Zugangsberechtigung, d.h. das Benutzerkennwort PW, eingegeben wird. Danach werden im Schritt 33 die Zahlen par(i), mit $0 \leq i \leq 16$, auf die vorstehend beschriebene Weise ermittelt, und gemäß den Blöcken 34 und 35 in Fig. 3A wird die Textlänge tl = 0 gesetzt bzw. wird die Anzahl L der verschlüsselten Zeichen bestimmt. Nach der Abfrage L > 0 ? bei 36, d.h. ob zu entschlüsselnde Zeichen vorliegen – wenn nicht, wird bei 37 zum Programm-Ende gegangen -, wird tl um 1 erhöht (tl = tl + 1, Schritt 38), und danach wird das erste (bzw. das nächste) zu entschlüsselnde Zeichen aus der „Codedatei", gegebenenfalls unter Datendekompression, in den Arbeitsspeicher eingelesen (Schritt 39 in Fig. 3A).

Um nun die „versteckten" Bits von vzbin und vcbin auslesen zu können, müssen ihre Stellen n(i) wieder festgestellt werden, wozu wiederum auf die beschriebene Weise die Permutation P – gleich wie beim Verschlüsseln – bestimmt wird, s. Feld 40 in Fig. 3A. Danach können die Bits von vcbin und vzbin ausgelesen werden (s. Felder 41 und 43 in Fig. 3A bzw. 3B), und es kann die Dezimalcodeform vc der Zufallszahl bestimmt werden (s. Feld 42 in Fig. 3B).

Auf gleiche Weise wie beim Verschlüsseln wird nun über die Funktion ALG1 die Zahl „code" abhängig von par(i) erzeugt, s. Feld 44, und in die Binärform (als „codebin") umgewandelt, s. Feld 45. Hiernach wird wieder über die XOR-Funktion die eigentliche Entschlüsselung gemäß

$$ASC(a\$)bin = vzbin \text{ XOR } codebin$$

vorgenommen, s. Feld 46 in Fig. 3B. Aus der Binärform ASC(a$)bin wird nun im Schritt 47 das ursprüngliche, unverschlüsselte Zeichen a$ bestimmt und gespeichert.

Danach wird bei 48 abgefragt, ob noch zu entschlüsselnde Zeichen vorliegen, und wenn ja, wird zu Schritt 38 (s. Fig. 3A) zurückgekehrt; wenn nein, ist das Programm-Ende 49 erreicht.

Verbale Beschreibung des Verschlüsselungs- bzw. Entschlüsselungsablaufes

(analog zum beigelegten Flußdiagramm)

I.) Verschlüsselung

1.) Zu Beginn des Programmes wird eine Paßworteingabe verlangt. Aus Gründen der Sicherheit und Praktikabilität werden 16 Zeichen gefordert. Diese Paßwortlänge ist jedoch nicht Pflicht sondern kann variabel gestaltet werden.

2.) Aus diesem Paßwort werden die Parameter abgeleitet. Dies sind Zahlen zwischen 0 und 1. Begründung: Durch die Kommazahlen ergibt sich ein relativ großer Spielraum für die Erzeugung von Algorithmen und Permutationen.

3.) Die Textlänge wird per Programm im Vorhinein auf Null gesetzt.

4.) Das Programm stellt nun fest aus wie vielen Zeichen (Bytes) die eingelesene Datei (Ursprungsdatei) besteht.

5.) Danach wird die Dateigröße abgefragt. Ist die Zahl der Zeichen > 0 wir das Programm fortgesetzt. Andernfalls wird es beendet.

6.) Der Computer erhält die Anweisung das tl-te Byte der Datei auszulesen. Dies geschieht in festgelegter Reihenfolge.

7.) Jedem ausgelesenen Zeichen wird dezimal ein ASC Wert zwischen 0 und 255 zugeordnet. Binär liegen diese Zahlen zwischen 00000000 und 11111111. Der ASC Wert wird in das Binärformat umgewandelt. Dieser Wert hat immer 8 Bit.

8.) Ein Zufallsgenerator erzeugt Zahlen zwischen 0 und 255. Diese Zahlen werden in weiterer Reihenfolge vc genannt und ebenfalls in das Binärformat umgewandelt (vcbin).

9.) Über einen Algorithmus, der von der Anzahl der eingelesenen Zeichen (tl) und von den Parametern des Paßwortes abhängig ist, wird aus vc (dezimal) eine Zahl erzeugt, die in weiterer Folge immer code genannt wird. Code (dezimal) wird ins Binärformat umgewandelt. Es entsteht codebin.

10.) Die Zahl vzbin wird erzeugt. Der binäre Wert des ASC Wertes des eingelesenen Zeichens ASC(a$)bin wird mit codebin XOR bitweise verschlüsselt. Es entsteht die 8 stellige binäre Zahl vzbin.

11.) Mit der Permutation wird begonnen. Diese bijektive Funktion ist ebenfalls von den Parametern und der Anzahl der eigelesenen Zeichen (tl) abhängig. Diese liest die einzelnen Bits von vcbin und vzbin aus und plaziert diese im Feld s an jene Stellen, die durch die Permutation festgelegt worden sind. Diese bedeutet folgendes: Das i-te Bit wird

an die Stelle n(i) geschrieben und an die Stelle n(i+16) das komplementäre Bit. Das Feld s besteht somit aus 32 Einsen und Nullen Bits. Es existieren immer genau 16 Einsen und Nullen Bits. Dadurch ist immer eine Unentscheidbarkeit gegeben. Es ist daher unmöglich zu wissen, an welchen Stellen die Zahlen vcbin und vzbin in diesem Feld gespeichert sind. Es sind immer alle Zahlenpaare in binärer Form möglich.

12.) Das Feld s wird bei der Komprimierung in 4 Teilbereiche aufgegliedert. Die ersten 8 Bit des Feldes werden dabei in einem ASC Wert (dezimal) umgewandelt. Es ergibt sich eine Zahl zwischen 0 und 255. Der Zeichencode dieser Zahl (CHR$) wird in der Codedatei gespeichert. Ebenso werden die 3 restlichen Bytes auf die gleiche Art und Weise gespeichert.

13.) Ein Abfragemechanismus überprüft, ob die eigelesene Textlänge tl kleiner als die Gesamtanzahl der Zeichen der Originaldatei ist. Tritt dieser Fall ein so kehrt das Programm zu Punkt 6 zurück. Im anderen Fall wird die Verschlüsselung beendet.

II.) Entschlüsselung:

1.) Das Paßwort wird abgefragt.

2.) Aus diesem Paßwort werden wieder die Parameter abgeleitet.

3.) Die Textlänge wird auf den Wert 0 gesetzt.

4.) Die Anzahl der zu entschlüsselnden Bytes der codierten Datei werden bestimmt. Diese Anzahl wird L genannt.

5.) Es erfolgt die Überprüfung ob L größer 0 ist. Bei positiver Überprüfung wird das Programm fortgesetzt andernfalls beendet.

6.) Die Textlänge tl wird jeweils um den Wert 1 gemäß dem Schema tl = tl +1 erhöht. Dadurch werden alle Zeichen der verschlüsselten Datei in festgelegter Reihenfolge ausgelesen.

7.) Nun werden die 4 Bytes aus der Codedatei gelesen. Diese 4 x 8 Bit repräsentieren jeweils ein Zeichen der usrprünglichen, unverschlüsselten Datei.

8.) Es erfolgt nun wieder die Permutation von natürlichen Zahlen. Diese liegen zwischen 1 und 32. Um eine korrekte Entschlüsselung zu ermöglichen muß es sich dabei natürlich um die gleiche Permutation wie bei der Verschlüsselung handeln !

9.) Die Einser- und Nuller – Bits von vcbin werden aus dem Feld s an den Stellen n(1) bis n(8) ausgelesen.

10.) Aus der Dezimalform von vc wird die Binärform vcbin ermittelt.

11.) Die Einser- und Nuller – Bits von vzbin werden aus dem Feld s an den Stellen 9 – 16 ausgelesen.

11.) Mittels eines Algorithmus wird nun die Zahl code erzeugt. Die Funktion, durch welche code erstellt wird ergibt sich aus den Parametern und ist weiters von vc und tl abhängig. Es ergibt sich die Zahl code in Dezimalform.

12.) Aus codedezimal wird nun codebin hergestellt.

13.) Der Wert ASC(a$)bin ergibt sich aus der XOR Verschlüsselung von vzbin und codebin. ASC(a$)bin stellt den ASC Wert eines Zeichens aus der Originaldatei dar.

14.) Das Originalzeichen a$ wird nun aus ASC(a$)bin bestimmt und in der Entschlüsselungsdatei gespeichert.

15.) Eine Abfrage kontrolliert, ob tl kleiner L/4 ist. Da die Datei in verschlüsselter Form 4 mal so lang ist wie die Ursprungsdatei muß hier die Abfrage nach L/4 erfolgen. Ist der endgültige Wert von tl erreicht ist die Entschlüsselung beendet. Ansonsten springt das Programm immer wieder zu Punkt 6 zurück und wiederholt den Auslese- bzw. Entschlüsselungsvorgang.

Bemerkungen zur Programmlogik sowie zum Programm selbst:

Kryptographie ist ein technischer Bereich, der immer mehr Einzug in das private Leben jedes einzelnen Mitbürgers nimmt. Das Bedürfnis nach Sicherheit und Privatsphäre nimmt zu. Unsere Programmlogik bzw. das Programm selbst soll dazu einen wichtigen Beitrag leisten.

Alle bisher bekannten Verschlüsselungssysteme basieren entweder auf einem „private-key" oder einem „public-key". Dies drückt aus, wie der Zugang zu einer verschlüsselten Information geregelt ist. Unsere Programmlogik läßt natürlich nur „private-key" Verschlüsselungen zu. Das bedeutet, daß sich Sender und Empfänger über genau dasselbe Paßwort einig sein müssen. Jeder andere, der im Besitz des Programmes, der verschlüsselten Datei und des Paßwortes ist, könnte sonst die verschlüsselte Datei knacken und den Originaltext wieder herstellen.
„Public-key" Programme können unseres Wissens nach nie so sicher gestaltet werden wie unser Programm . Deshalb wäre trotzdem möglich für den ersten Datenaustausch ein „public-key" Programm, beispielsweise auf RSA-Basis zu verwenden und danach auf unser Programm umzusatteln.

Nach unseren Informationen basieren alle Verschlüsselungsprogramme im wesentlichen auf einer 64, 128, 256,..... Bit Verschlüsselung. Das bedeutet, daß die Entschlüsselung der Nachricht mit zunehmender Anzahl der Bits schwieriger wird. Dies vor allem deshalb, weil die dazu erforderliche Rechenleistung erst einmal zur Verfügung gestellt werden muß.

Nach Informationen aus dem universitären Bereich, werden dort verwendete Großrechner im Jahr 2000 durchschnittlich 3,2 Milliarden Codes pro Sekunde austesten können. Dies wird neuerlich zu einer Steigerung der Bitzahl bei den gängigen Verschlüsselungsprogrammen führen.

Diese Steigerung geht natürlich nicht nur mit einem höheren Zeitaufwand für die Entschlüsselung einher sondern natürlich auch mit einem deutlich gesteigerten Zeitbedarf bei der Verschlüsselung. Besonders Privatanwender mit älteren Computersystem fällt diese Tatsache schmerzlich auf. Geringere Rechenzeiten sind wieder mit Herabsetzung der Bitanzahl möglich. Daraus folgt wieder geringere Sicherheit.

Primär haben wir unsere Programmlogik und in weiterer Folge unser erstes Programm mit dem Marktführer PGP verglichen. PGP steht für „Pretty Good Privacy" und stellt, was das Verschlüsseln von kleineren Dateien und das Verschicken von e-mails darstellt nach Aussagen verschiedenster Experten gegenwärtig den Marktführer und momentanen technischen Standard auf dem privaten Bereich dar.

Neben verschiedenen Verschlüsselungssystemen wie beispielsweise DES geht PGP von RSA aus. RSA benötigt zur Verschlüsselung einer gleichen Datenmenge ungefähr 100 mal länger als DES. Zur Erhöhung der Sicherheit wurden Varianten wie „Triple-DES" usw. geschaffen. Der Erfinder von PGP (Philip Zimmerman) schlägt für sein PGP gegenwärtig eine 1024 Bit Verschlüsselung vor. Es ergibt sich das bereits oben angesprochene Problem. Gesteigerte Sicherheit ist möglich, jedoch nur unter Inkaufnahme deutlich gesteigerter Rechenzeiten.

Unsere Programmlogik bzw. ein darauf basierendes Computerprogramm beheben diesen Makel.

Wir schaffen es erhöhte Sicherheit bei deutlich gesteigerter Geschwindigkeit für Ver- und Entschlüsselungsprozeß zu bieten. Das ist die Quintessenz des Programmes.

Diese Tatsache beruht vor allem auf zwei Kernpunkten, die unsere Programmlogik auszeichnen und von anderen Verschlüsselungssystemen unterscheiden.

1.) Unsere Programmlogik basiert nicht auf einer konstanten Schlüssellänge. Man kann nicht behaupten, daß es sich dabei um eine 64, 128, usw. Bit-Verschlüsselung handelt. Der Schlüssel wird bei unserem Programm durch die Dateilänge und den Zufall bestimmt.Die so geschaffene Variabilität führt zu immer neuen verschiedenen Mustern. Dabei handelt es sich im Programm einstweilen nur um einen Vorschlüssel, der im Programm zu einem späteren Zeitpunkt noch transformiert wird. Man kann also behaupten, daß bei unserem Programm die Schlüssellänge unendlich ist. Wie in den zusätzlich beigelegten Schriften angeführt, kann der Schlüssel beispielsweise durch das weiße Rauschen einer Diode generiert werden.

2.) Unsere Programmlogik stellt einen Unbefugten, der versucht die Datei zu entschlüsseln, vor den Grundsatz der Unentscheidbarkeit. Dies bedeutet, daß bei der Entschlüsselung der verschlüsselten Datei immer alle Varianten möglich sind. Dies schließt alle „sinnvollen" als auch „nicht sinnvollen" Varianten ein. Bei konventionellen Verschlüsselungssystem weiß man, daß man sobald zumindest ein Teil des Textes eine sinnvolle Bedeutung ergibt, auf der richtigen Spur ist. Dies ist bei unserem Programm nicht möglich, da immer alle Varianten gleich wahrscheinlich sind. Es ergibt sich die Unentscheidbarkeit.
Selbst wenn der Betrachter nun den richtig entschlüsselten Text vor sich hätte , würde er nicht wissen, daß dieser Text richtig ist. Das ist eine ganz wichtige Kernaussage unseres Programmes.

Egal wie schnell nun die Großrechner einmal werden sollten, diese Unentscheidbarkeit wird immer gegeben bleiben. Ein Unbefugter, der nicht weiß was richtig oder falsch ist kann mit einer Datei nichts anfangen.

Ein Programm mit unserer Programmlogik bietet auch für den Privatbenutzer mit einem veralteten Rechnersystem bestmögliche Sicherheit bei geringem Zeitaufwand. Aus privaten Versuchen wissen wir, daß die Verschlüsselung von einer Datei mit der Größe von
1 MB ca. 40 Sekunden dauert. Dabei handelt es sich noch um ein nicht optimiertes und unausgereiftes Programm. Alle uns bekannten anderen Verschlüsselungssysteme benötigen ein vielfaches dieser Zeit.

Aus dieser einfachen und sehr schnellen Programmlogik ergeben sich natürlich eine Fülle von Möglichkeiten, da der Einsatz von wirksamen Verschlüsselungssystemen bisher allzu oft an der dafür benötigten Zeit für die Entschlüsselung gescheitert ist. Wir haben unser Programm primär für die persönliche Ver- und Entschlüsselung von Textdateien und e-mails gestaltet. Wie oben angeführt sind aber auch Dateien von 1 MB kein Problem und es gelang uns sogar ganze Festplatten zu ver- und entschlüsseln.

Wir können uns vorstellen, daß unsere Programmlogik nicht zuletzt auch auf Grund Ihrer Einfachheit Einzug in die verschiedensten technischen Bereiche erhält. Es ist ja die Ver- und Entschlüsselung von sämtlichen Daten möglich.

Ein möglicher Bereich wäre beispielsweise das digitale Fernsehen. Bis dato sind alle verwendeten Verschlüsselungssysteme geknackt worden. Unsere schnelle Programmlogik könnte dem einen Riegel vorsetzen.

Bei entsprechend angepaßter Hardware sollte Echtzeitentschlüsselung möglich sein. Für den Einzelfall müßten genauere Prüfungen angesetzt werden.
Digitale Telephonie ist ein weiteres Stichwort. Der Einsatzbereich ist wirklich universell und kann quasi allen technischen Bedürfnissen angepaßt werden.

Ein weiteres interessantes Merkmal, daß sich durch unsere Programmlogik ergibt ist die Tatsache der digitalen Signatur. Eine verschlüsselte Datei kann nicht abgefangen und verändert werden ohne das der Empfänger dies bemerkt. Der Austausch eines einzigen Bits hat bereits Folgen auf die Entschlüsselung. Funktioniert die Entschlüsselung beim Empfänger unter Einhaltung der vorgegebenen Parameter nicht, so weiß man, daß mit Sicherheit an der verschlüsselten Datei manipuliert wurde.
Dabei geht es vor allem auch darum Dateien für sich selbst zu sichern, also ein Paßwort zu verwenden, daß sonst niemand kennt.

Abschlußbemerkungen:

Nach eingehender Recherche in den Datenbanken der verschiedensten Patentamte weltweit konnten wir persönlich keine gleich geartete Programmlogik entdecken. Auch das Studium zusätzlicher Literatur und fieberhafte Suche im Internet brachte nichts zu Tage. Es gibt sehr wohl Teilbereiche innerhalb unseres Programmes, die auf bereits dagewesenen Verfahren beruhen, doch die obig angeführten Kernpunkte glauben wir für neu und noch nie dagewesen bezeichnen zu dürfen. Dies bezieht sich insbesondere auf die zu schützende Programmlogik.
Wir hoffen damit einen neuen und entscheidenden Schritt auf dem Gebiet der Kryptographie getan zu haben.

Ansprüche

1. Verfahren zum Verschlüsseln von Daten, die aus einer Anzahl von Zeichen bestehen, unter Verwendung eines Rechners, dadurch gekennzeichnet, dass für jedes Zeichen eine Zufallszahl generiert wird, aus der mittels einer vorgegebenen, injektiven Funktion eine Codezahl erzeugt wird, mit der das jeweilige Zeichen gemäß der EXKLUSIV-ODER-Funktion zu einem verschlüsselten Zeichen logisch verknüpft wird, und dass die verschlüsselten Zeichen zusammen mit den zugehörigen Zufallszahlen in binärer Form jeweils in einem Feld gespeichert bzw. übertragen werden, das eine Feldlänge aufweist, die doppelt so groß ist wie die Zahl der Bits der jeweiligen verschlüsselten Zeichen und der zugehörigen Zufallszahlen zusammen, wobei die restlichen Plätze des jeweiligen Feldes mit den zu den Bits der verschlüsselten Zeichen und Zufallszahlen komplementären Bits aufgefüllt werden.

2. Verfahren nach Anspruch 1, dadurch gekennzeichnet, dass die Plätze des Feldes für die Bits des jeweiligen verschlüsselten Zeichens und der jeweiligen Zufallszahl auf Basis einer vorgegebenen bijektiven Funktion, verschieden von der ursprünglichen Reihung der Bits, festgelegt werden.

3. Verfahren nach Anspruch 2, dadurch gekennzeichnet, dass als bijektive Funktion eine von einem eingegebenen Benutzerkennwort abhängige Funktion vorgegeben ist.

4. Verfahren nach einem der Ansprüche 1 bis 3, dadurch gekennzeichnet, dass die vorgegebene injektive Funktion für die Erzeugung der Codezahlen als von der Anzahl der Zeichen der zu verschlüsselnden Daten abhängige Funktion festgelegt wird.

5. Verfahren nach einem der Ansprüche 1 bis 4, dadurch gekennzeichnet, dass die vorgegebene injektive Funktion für die Erzeugung der Codezahlen von einem eingegebenen Benutzerkennwort abhängig festgelegt wird.

6. Verfahren nach einem der Ansprüche 1 bis 5, dadurch gekennzeichnet, dass die generierten Zufallszahlen natürliche Zahlen zwischen 0 und 255 sind.

7. Verfahren zum Entschlüsseln von mit einem Verfahren nach einem der Ansprüche 1 bis 6 verschlüsselten Daten, dadurch gekennzeichnet, dass aus den zusammen mit den verschlüsselten Zeichen übermittelten Zufallszahlen mittels der selben vorgegebenen injektiven Funktion die jeweiligen Codezahlen erzeugt werden, die dann mit den zugehörigen verschlüsselten Zeichen über die EXKLUSIV-ODER-Funktion logisch verknüpft werden, um die ursprünglichen Zeichen zu erhalten.

8. Verfahren nach Anspruch 7, zum Entschlüsseln von mit einem Verfahren nach Anspruch 2 oder 3 verschlüsselten Daten, dadurch gekennzeichnet, dass unter Anwendung der vorgegebenen bijektiven Funktion die in den jeweiligen Feldern in unterschiedlichen Reihungen enthaltenen Bits der jeweiligen verschlüsselten Zeichen und zugehörigen Zufallszahlen in der ursprünglichen Reihung ermittelt werden, bevor die Codezahlen erzeugt und mit den verschlüsselten Zeichen logisch verknüpft werden.

9. Einrichtung zum Verschlüsseln von Daten, die aus einer Anzahl von Zeichen bestehen, mit einem Rechner, dadurch gekennzeichnet, dass der Rechner eingerichtet ist, für jedes Zeichen eine Zufallszahl zu generieren, aus der mittels einer vorgegebenen, injektiven Funktion eine Codezahl erzeugt wird, mit der das jeweilige Zeichen gemäß der EXKLUSIV-ODER-Funktion zu einem verschlüsselten Zeichen logisch verknüpft wird, und die verschlüsselten Zeichen zusammen mit den zugehörigen Zufallszahlen in binärer Form jeweils in einem Feld zu speichern bzw. zu übertragen, das eine Feldlänge aufweist, die doppelt so groß ist wie die Zahl der Bits der jeweiligen verschlüsselten Zeichen und der zugehörigen Zufallszahlen zusammen, wobei die restlichen Plätze des jeweiligen Feldes mit den zu den Bits der verschlüsselten Zeichen und Zufallszahlen komplementären Bits aufgefüllt werden.

10. Einrichtung nach Anspruch 9, dadurch gekennzeichnet, dass der Rechner eingerichtet ist, die Plätze des Feldes für die Bits des jeweiligen verschlüsselten Zeichens und der jeweiligen Zufallszahl auf Basis einer vorgegebenen bijektiven Funktion, unterschiedlich von der ursprünglichen Reihung der Bits, festzulegen.

11. Einrichtung nach Anspruch 10, dadurch gekennzeichnet, dass der Rechner eingerichtet ist, als bijektive Funktion eine von einem eingegebenen Benutzerkennwort abhängige Funktion vorzugeben.

12. Einrichtung nach einem der Ansprüche 9 bis 11, dadurch gekennzeichnet, dass der Rechner eingerichtet ist, die vorgegebene injektive Funktion für die Erzeugung der Codezahlen als von der Anzahl der Zeichen der zu verschlüsselnden Daten abhängige Funktion festzulegen.

13. Einrichtung nach einem der Ansprüche 9 bis 12, dadurch gekennzeichnet, dass der Rechner eingerichet ist, die vorgegebene injektive Funktion für die Erzeugung der Codezahlen von einem eingegebenen Benutzerkennwort abhängig festzulegen.

14. Einrichtung nach einem der Ansprüche 9 bis 13, dadurch gekennzeichnet, dass die generierten Zufallszahlen natürliche Zahlen zwischen 0 und 255 sind.

15. Einrichtung zum Entschlüsseln von mit einer Einrichtung nach einem der Ansprüche 9 bis 14 verschlüsselten Daten, mit einem Rechner, dadurch gekennzeichnet, dass der Rechner eingerichtet ist, aus den zusammen mit den verschlüsselten Zeichen übermittelten Zufallszahlen mittels der selben vorgegebenen injektiven Funktion die jeweiligen Codezahlen zu erzeugen, die dann mit den zugehörigen verschlüsselten Zeichen über die EXKLUSIV-ODER-Funktion logisch verknüpft werden, um die ursprünglichen Zeichen zu erhalten.

16. Einrichtung nach Anspruch 15, dadurch gekennzeichnet, dass der Rechner eingerichtet ist, unter Anwendung der vorgegebenen bijektiven Funktion die in den jeweiligen Feldern in unterschiedlichen Reihungen enthaltenen Bits der jeweiligen verschlüsselten Zeichen und zugehörigen Zufallszahlen in der ursprünglichen Reihung zu ermitteln, bevor die Codezahlen erzeugt und mit den verschlüsselten Zeichen logisch verknüpft werden.

17. Maschinenlesbarer Programmdatenträger mit einem Computerprogramm, das zur Durchführung eines Verfahrens nach einem der Ansprüche 1 bis 8 eingerichtet ist.

FIG. 1A

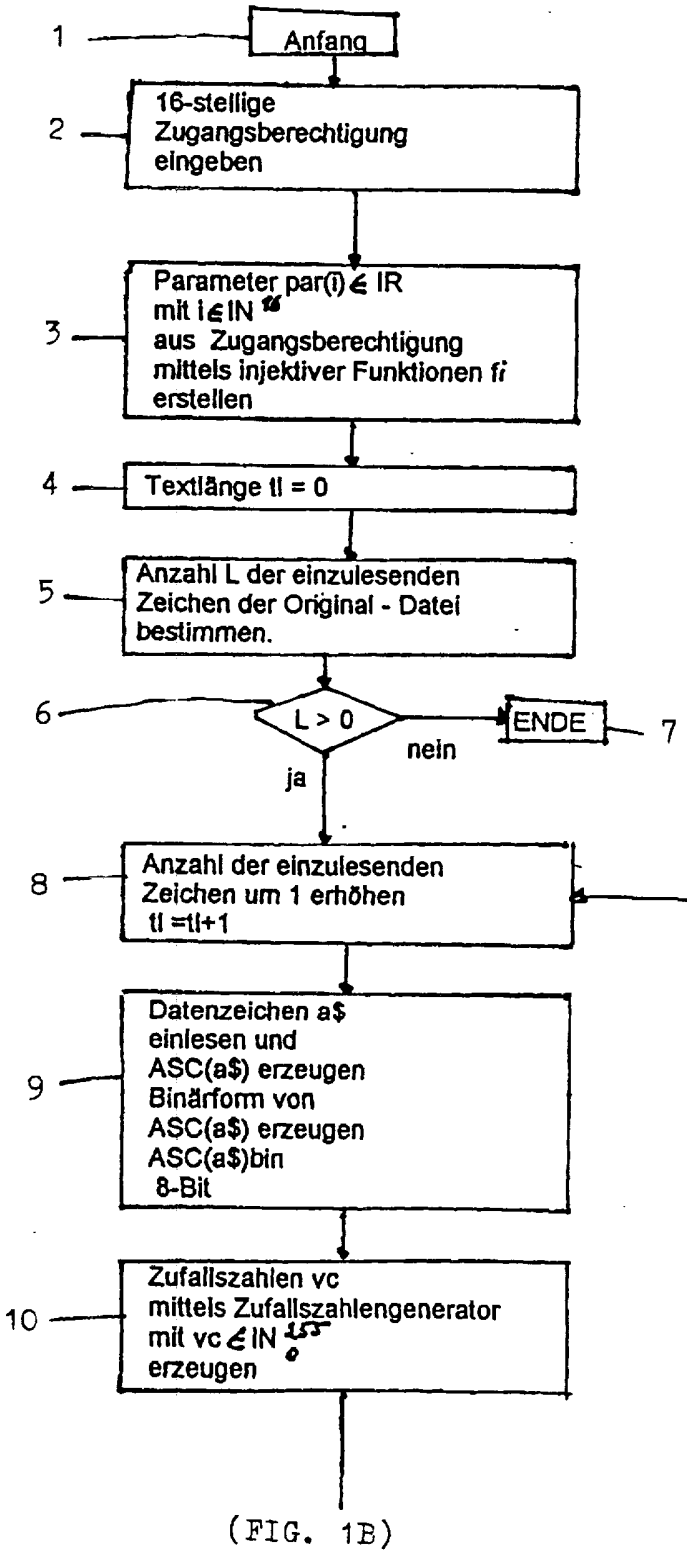
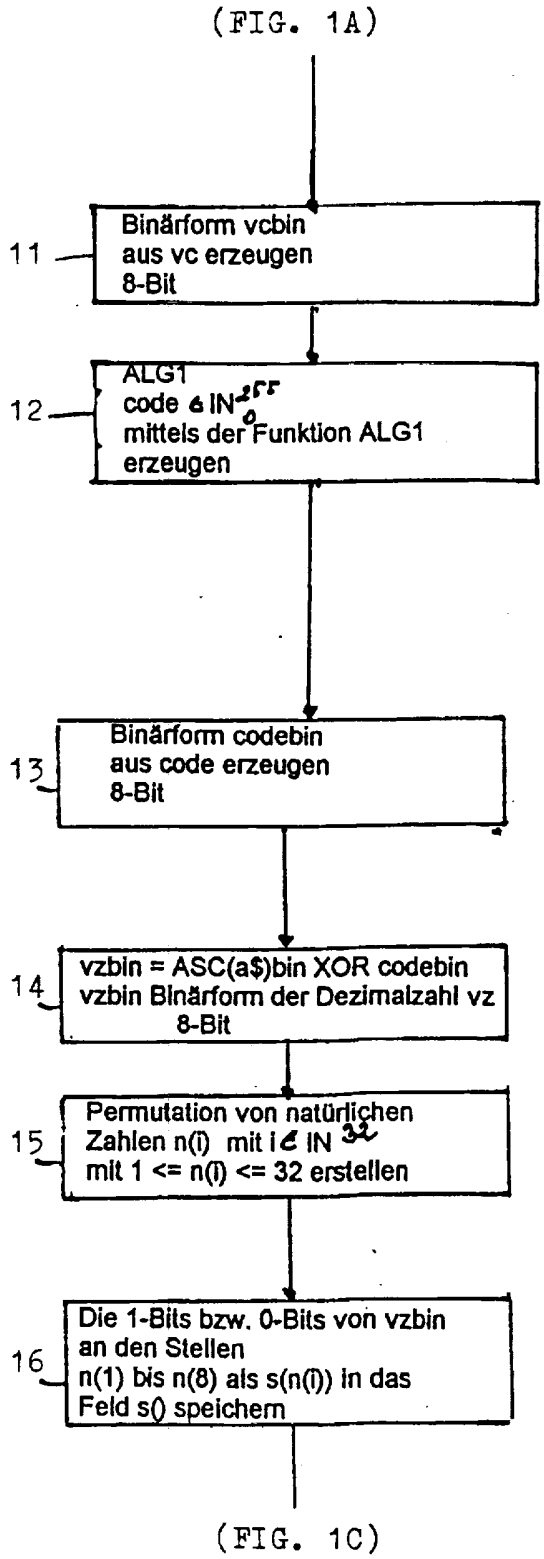

1
Anfang
2
16-stellige Zugangsberechtigung eingeben
3
Parameter par(i) ∈ IR mit i ∈ IN aus Zugangsberechtigung mittels injektiver Funktionen fi erstellen
4
Textlänge tl = 0
5
Anzahl L der einzulesenden Zeichen der Original - Datei bestimmen.
6
L > 0
nein
ja
ENDE
7
8
Anzahl der einzulesenden Zeichen um 1 erhöhen tl =tl+1
9
Datenzeichen a$ einlesen und ASC(a$) erzeugen Binärform von ASC(a$) erzeugen ASC(a$)bin 8-Bit
10
Zufallszahlen vc mittels Zufallszahlengenerator mit vc ∈ IN erzeugen
(FIG. 1B)
(FIG. 1B)

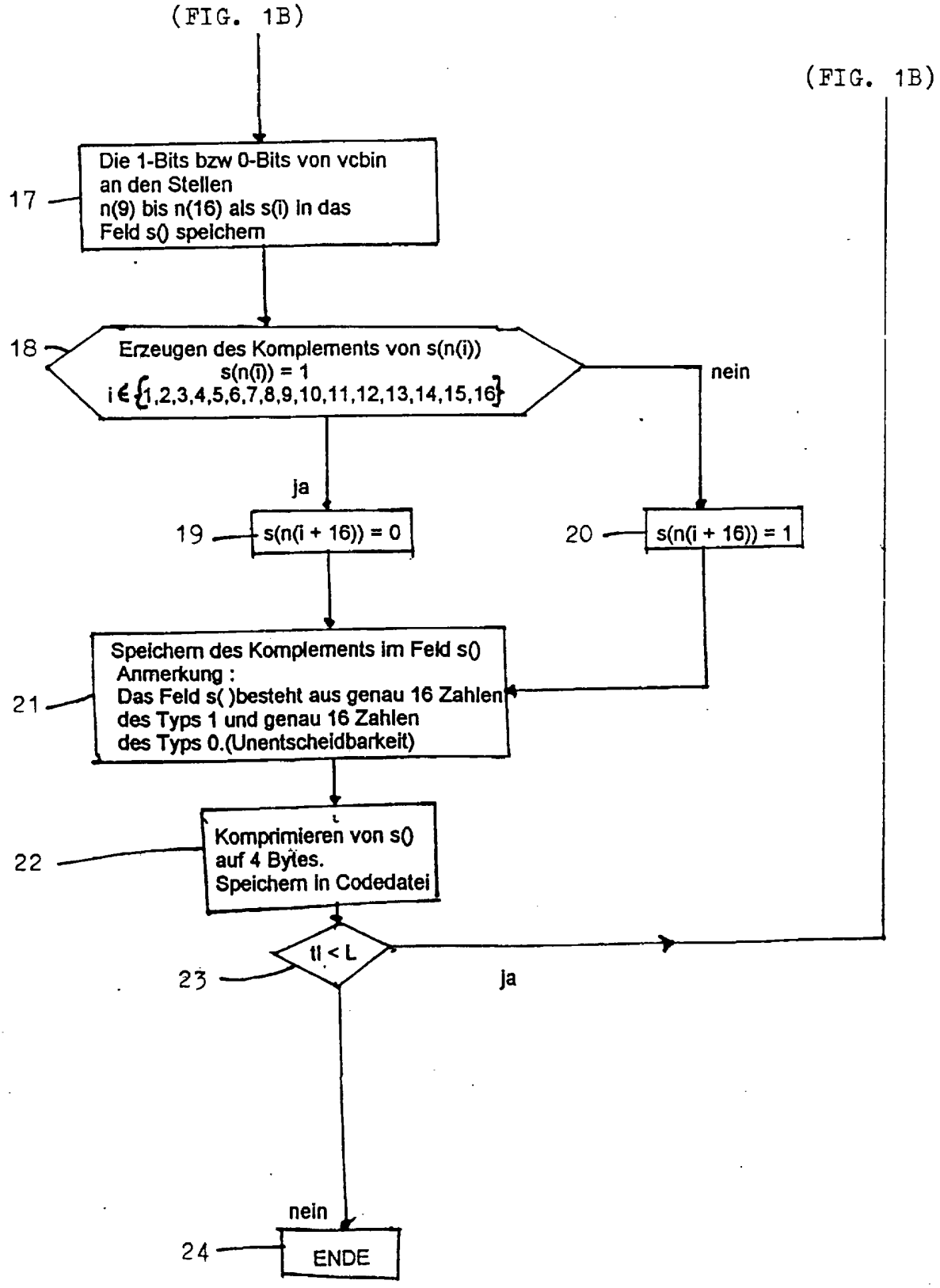
FIG. 1B



(FIG. 1A)

(FIG. 1C)

FIG. 1C


(FIG. 1B)
(FIG. 1B)
17
Die 1-Bits bzw 0-Bits von vcbin an den Stellen n(9) bis n(16) als s(i) in das Feld s() speichern
18
Erzeugen des Komplements von s(n(i))
s(n(i)) = 1
i ∈ {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}
nein
ja
19
s(n(i + 16)) = 0
20
s(n(i + 16)) = 1
21
Speichern des Komplements im Feld s()
Anmerkung :
Das Feld s( )besteht aus genau 16 Zahlen des Typs 1 und genau 16 Zahlen des Typs 0.(Unentscheidbarkeit)
22
Komprimieren von s() auf 4 Bytes.
Speichern in Codedatei
23
tl < L
ja
nein
24
ENDE

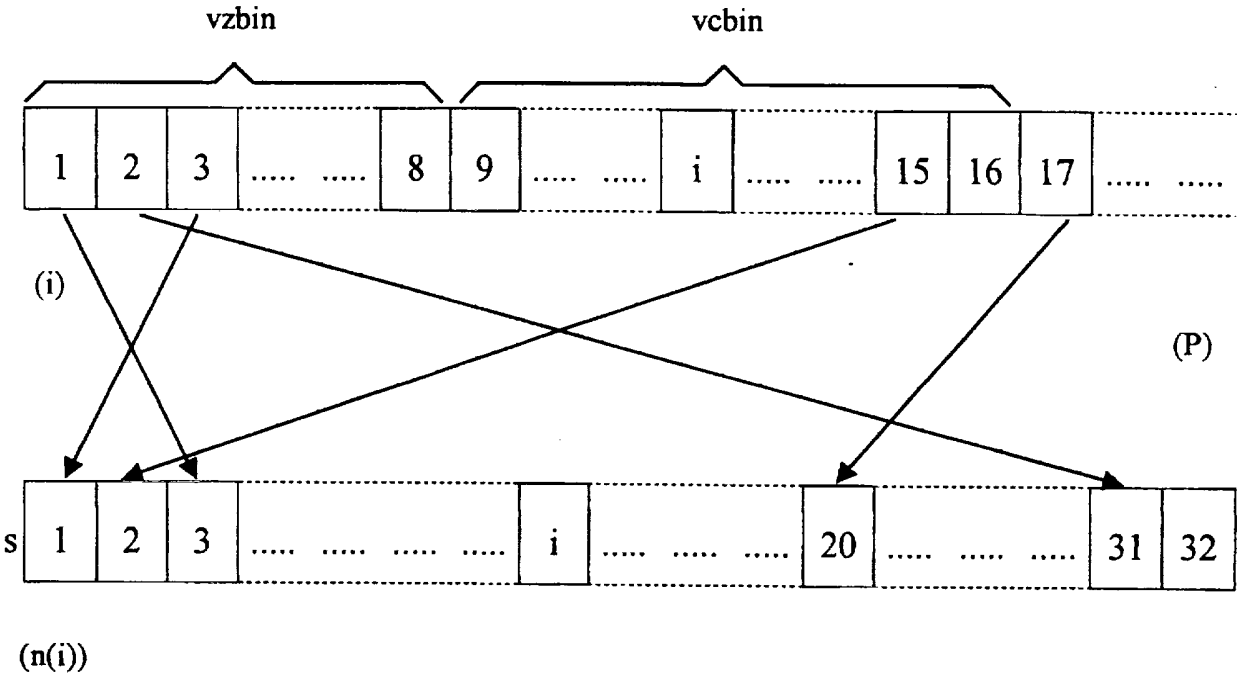


FIG. 2

FIG. 3A

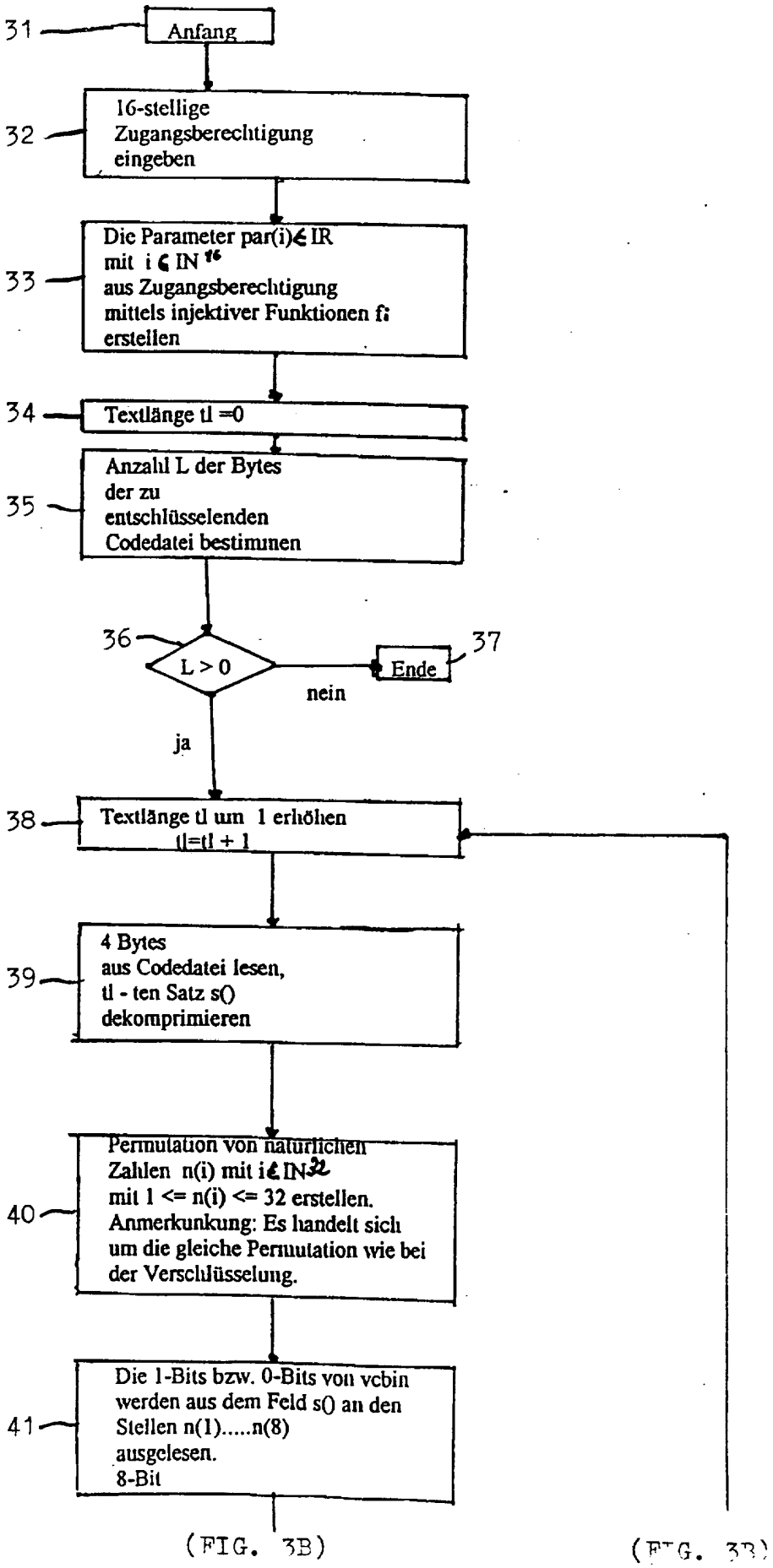
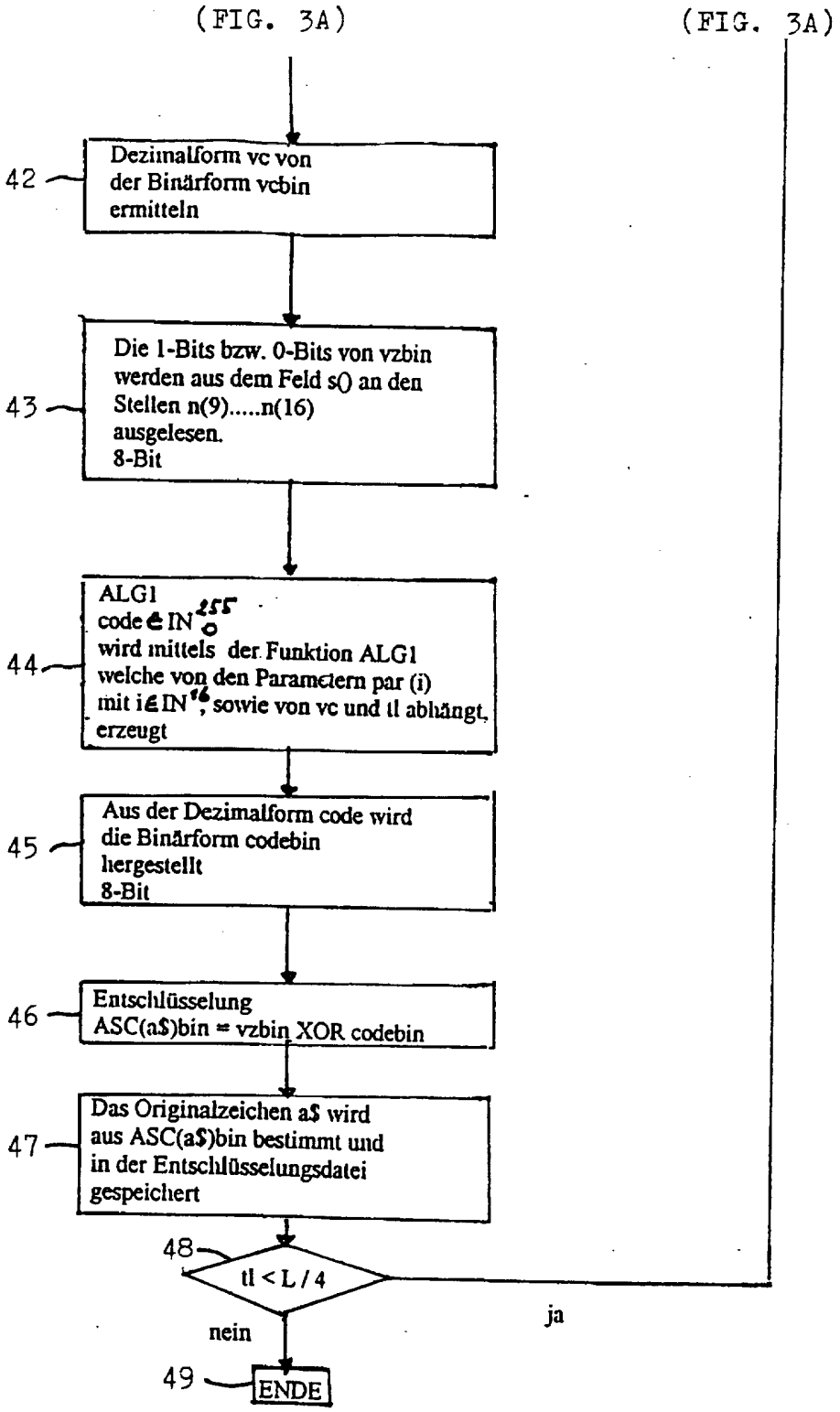

31
Anfang
32
16-stellige Zugangsberechtigung eingeben
33
Die Parameter par(i)∈ IR mit i ∈ IN 16 aus Zugangsberechtigung mittels injektiver Funktionen fi erstellen
34
Textlänge tl =0
35
Anzahl L der Bytes der zu entschlüsselenden Codedatei bestimmen
36
L > 0
37
Ende
nein
ja
38
Textlänge tl um 1 erhöhen tl=tl + 1
39
4 Bytes aus Codedatei lesen, tl - ten Satz s() dekomprimieren
40
Permutation von natürlichen Zahlen n(i) mit i∈ IN32 mit 1 <= n(i) <= 32 erstellen. Anmerkunkung: Es handelt sich um die gleiche Permutation wie bei der Verschlüsselung.
41
Die 1-Bits bzw. 0-Bits von vcbin werden aus dem Feld s() an den Stellen n(1).....n(8) ausgelesen. 8-Bit
(FIG. 3B)
(FIG. 3B)

FIG. 3B


(FIG. 3A)
(FIG. 3A)
42
Dezimalform vc von der Binärform vcbin ermitteln
43
Die 1-Bits bzw. 0-Bits von vzbin werden aus dem Feld s() an den Stellen n(9).....n(16) ausgelesen.
8-Bit
44
ALG1
code ∈ IN 255 0
wird mittels der Funktion ALG1 welche von den Parametern par (i) mit i ∈ IN 16, sowie von vc und tl abhängt, erzeugt
45
Aus der Dezimalform code wird die Binärform codebin hergestellt
8-Bit
46
Entschlüsselung
ASC(a$)bin = vzbin XOR codebin
47
Das Originalzeichen a$ wird aus ASC(a$)bin bestimmt und in der Entschlüsselungsdatei gespeichert
48
tl < L / 4
ja
nein
49
ENDE

ÖSTERREICHISCHES PATENTAMT
A-1014 Wien, Kohlmarkt 8-10, Postfach 95
TEL. +43/(0)1/53424; FAX +43/(0)1/53424-535; TELEX 136847 OEPA A
Postscheckkonto Nr. 5.160.000; UID-Nr. ATU38266407; DVR: 0078018

RECHERCHENBERICHT

zu 15 GM 715/99
Ihr Zeichen:

Klassifikation des Antragsgegenstandes gemäß IPC[7] : G06F 7/58, H04L 9/20, H04L 9/18,

Recherchierter Prüfstoff (Klassifikation): G06F, H04L

Konsultierte Online-Datenbank: EPODOC, WPI, PAJ, ACM Digital Library

Die nachstehend genannten Druckschriften können in der Bibliothek des Österreichischen Patentamtes während der Öffnungszeiten (Montag bis Freitag von 8 - 12 Uhr 30, Dienstag 8 bis 15 Uhr) unentgeltlich eingesehen werden. Bei der von der Hochschülerschaft TU Wien Wirtschaftsbetriebe GmbH im Patentamt betriebenen Kopierstelle können schriftlich (auch per Fax. Nr. 01 / 533 05 54) oder telefonisch (Tel. Nr. 01 / 534 24 - 153) **Kopien** der ermittelten Veröffentlichungen bestellt werden.
Auf Anfrage gibt das Patentamt Teilrechtsfähigkeit (TRF) gegen Entgelt zu den im Recherchenbericht genannten Patentdokumenten allfällige veröffentlichte „Patentfamilien" (denselben Gegenstand betreffende Patentveröffentlichungen in anderen Ländern, die über eine gemeinsame Prioritätsanmeldung zusammenhängen) bekannt. Diesbezügliche Auskünfte erhalten Sie unter der Telefonnummer 01 / 534 24 - 725.

Kategorie	Bezeichnung der Veröffentlichung (Ländercode, Veröffentlichungsnummer, Dokumentart (Anmelder), Veröffentlichungsdatum, Textstelle oder Figur (soweit erforderlich)	Betreffend Anspruch
A	EP 877 509 A2 (IBM), 11.11.1998 * gesamtes Dokument *	1-17
A	US 5,841,872 A (COLVIN), 24.11.1998 * gesamtes Dokument *	1-17
A	LEIBERICH Otto.'Vom diplomatischen Code zur Falltürfunktion'. In: Spektrum der Wissenschaft, Juni 1999, S. 26 - 29. * S.30, 3. Spalte über Wurm- oder Strom-Chiffrierverfahren *	1-17

☒ Fortsetzung siehe Folgeblatt

Kategorien der angeführten Dokumente (dient in Anlehnung an die Kategorien bei EP- bzw. PCT-Recherchenberichten nur **zur raschen Einordnung** des ermittelten Stands der Technik, stellt keine Beurteilung der Erfindungseigenschaft dar):
„**A**" Veröffentlichung, die den **allgemeinen Stand der Technik** definiert.
„**Y**" Veröffentlichung von Bedeutung; die Erfindung kann nicht als neu (bzw. auf erfinderischer Tätigkeit beruhend) betrachtet werden, wenn die Veröffentlichung mit einer oder mehreren weiteren Veröffentlichungen dieser Kategorie in Verbindung gebracht wird und diese **Verbindung für den Fachmann naheliegend** ist.
„**X**" Veröffentlichung **von besonderer Bedeutung**; die Erfindung kann allein aufgrund dieser Druckschrift nicht als neu (bzw. auf erfinderischer Tätigkeit beruhend) angesehen werden.
„**P**" zwischenveröffentlichtes Dokument von besonderer Bedeutung (**älteres Recht**)
„**&**" Veröffentlichung, die Mitglied derselben **Patentfamilie** ist.

Ländercodes:
AT = Österreich; **AU** = Australien; **CA** = Kanada; **CH** = Schweiz; **DD** = ehem. DDR; **DE** = Deutschland; **EP** = Europäisches Patentamt; **FR** = Frankreich; **GB** = Vereinigtes Königreich (UK); **JP** = Japan; **RU** = Russische Föderation; **SU** = ehem. Sowjetunion; **US** = Vereinigte Staaten von Amerika (USA); **WO** = Veröffentlichung gem. PCT (WIPO/OMPI); weitere siehe WIPO-Appl. Codes

Datum der Beendigung der Recherche: 25.8.2000 Prüferin: Fastenbauer

ÖSTERREICHISCHES PATENTAMT
A-1014 Wien, Kohlmarkt 8-10, Postfach 95
TEL. +43/(0)1/53424; FAX +43/(0)1/53424-535; TELEX 136847 OEPA A
Postscheckkonto Nr. 5.160.000; UID-Nr. ATU38266407; DVR: 0078018

Folgeblatt zu GM 715/99

Kategorie	Bezeichnung der Veröffentlichung (Ländercode, Veröffentlichungsnummer, Dokumentart (Anmelder), Veröffentlichungsdatum, Textstelle oder Figur (soweit erforderlich)	Betreffend Anspruch
A	TANG Lei.'Methods for Encrypting and Decrypting MPEG Video Data Efficiently'. IN: Proceedings of the fourth ACM International multimedia conference on Proceedings ACM Multimedia 96. Nov. 1996, Boston * gesamtes Dokument*	1-17
A	KLEIN Shmuel T., BOOKSTEIN Abraham, DEERWESTER Scott. 'Storing Text Retrieval Systems on CD-ROM: Compression and Encryption Considerations'.IN: ACM Transactions on Information Systems, Vol.7, No.3, Juli 1989, S.230-245. * gesamtes Dokument *	1-17

☐ Fortsetzung siehe Folgeblatt