(19) **Bundesrepublik Deutschland**
Deutsches Patent- und Markenamt

(10) **DE 697 24 946 T2** 2004.08.12

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 0 895 148 B1**
(21) Deutsches Aktenzeichen: **697 24 946.8**
(96) Europäisches Aktenzeichen: **97 113 262.6**
(96) Europäischer Anmeldetag: **31.07.1997**
(97) Erstveröffentlichung durch das EPA: **03.02.1999**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **17.09.2003**
(47) Veröffentlichungstag im Patentblatt: **12.08.2004**

(51) Int Cl.[7]: **G06F 1/00**

(73) Patentinhaber:
Siemens AG, 80333 München, DE

(72) Erfinder:
Benson, Dr., Glenn, 81739 München, DE

(84) Benannte Vertragsstaaten:
DE, FR, GB, IT

(54) Bezeichnung: **Programmvermietungssystem und Verfahren zur Vermietung von Programmen**

Anmerkung: Innerhalb von neun Monaten nach der Bekanntmachung des Hinweises auf die Erteilung des europäischen Patents kann jedermann beim Europäischen Patentamt gegen das erteilte europäische Patent Einspruch einlegen. Der Einspruch ist schriftlich einzureichen und zu begründen. Er gilt erst als eingelegt, wenn die Einspruchsgebühr entrichtet worden ist (Art. 99 (1) Europäisches Patentübereinkommen).

Die Übersetzung ist gemäß Artikel II § 3 Abs. 1 IntPatÜG 1991 vom Patentinhaber eingereicht worden. Sie wurde vom Deutschen Patent- und Markenamt inhaltlich nicht geprüft.

Beschreibung

[0001] Die meisten Computerprogramme sind mit wenigen Ausnahmen Instantiierungen geistigen Eigentums und laufen nach Installation auf Abruf ab. Normalerweise kann man eine kostengünstige Kopie eines Computerprogramms durch wenig mehr als einfache Ausführung eines Kopierbefehls aufbauen und installieren. Andererseits läßt sich ein materielles Besitzstück wie beispielsweise ein Kraftfahrzeug nicht leicht kopieren. Es ist infolgedessen viel leichter, materielle Besitzstücke zu vermieten als Computerprogramme. Bei einem materiellen Besitzstück bezahlt der Mieter zuerst eine Mietgebühr und nimmt dann das Besitzstück physisch in Besitz. Am Ende der Mietzeit gibt der Mieter das materielle Besitzstück dem Eigentümer zurück. Bei Software ist es andererseits ziemlich sinnlos, daß der Kunde das Programm dem Eigentümer zurückgibt, da nicht garantiert werden kann, daß sich der Kunde nicht seine eigene Sicherungskopie angeeignet hat. In der Abwesenheit ausreichender Sicherheitsmaßnahmen könnte ein in der Rolle eines Angreifers handelnder Kunde möglicherweise die Software eine kurze Zeit lang mieten und danach die angeeignete Sicherungskopie ohne Bezahlung weiterer Mietgebühren benutzen.

[0002] In der vorliegenden Erfindung wird Softwaremiete als Computersystem und -verfahren definiert, mit dem Datensätze zur Miete (Benutzung) sicher gespeichert werden. Man betrachte beispielsweise die Metrik der Benutzungszeit-Miete. Wenn der Kunde die gemietete Software eine Stunde lang am ersten Tag und zwei Stunden am zweiten Tag ablaufen läßt, dann zeigen die gesicherten Auditwege eine Stunde am Ende des ersten Tages und drei Stunden am Ende des zweiten Tages. Sichere Softwaremiete bedeutet, daß ein Kunde die Systemsicherheit nicht durch Löschen, Ersetzen oder Abändern von Auditwegen hintergehen kann. Normalerweise überwacht die Software fortlaufend die Auditwege, um festzustellen, wann ein Schwellwert überschritten wird. Wenn daher die beispielhafte Software einen Schwellwert von fünf Stunden aufweist, dann kann der Kunde die Software noch zwei Stunden lang ablaufen lassen, und dann hält sie an. Ein weiterer beispielhafter Schwellwert ist, wieviele Male insgesamt die Software gefahren werden kann.

[0003] Es gibt gegenwärtig Mietmechanismen, die alle oben aufgeführten Eigenschaften der vorliegenden Erfindung aufweisen, z. B. Dongles [1]. Dongles weisen einen nichtflüchtigen Speicher auf, der durch Paßworte geschützt sein kann. Dieser paßwortgeschützte Speicher kann möglicherweise für Softwaremiete benutzt werden. Es ist eine Eigenschaft dieses Mietmechanismus, daß er die Unterstützung einer gesicherten Mietvorrichtung, z. B. eines Dongles erfordert. Die gesicherte Mietvorrichtung enthält gesicherte aktualisierbare Speicherstellen (SUSL – Secured Updateable Storage Locations), in denen Informationen bezüglich der Nutzung der gemieteten Software aufgezeichnet sind. Jede SUSL hat die Eigenschaft, daß die SUSL auf einer gesicherten Vorrichtung residiert und Schutz gegen Angriff bietet. Normalerweise ist mindestens eine SUSL für jede Einheit gemieteter Software erforderlich. Wenn beispielsweise ein Kunde ein Textverarbeitungsprogramm, ein Tabellenkalkulationsprogramm und ein Spielprogramm mietet, dann muß (müssen) die Mietvorrichtung(en) mindestens drei SUSL bereitstellen. Diese SUSL sind relativ teuer und im Vergleich mit anderer Speicherung auf der Kundenmaschine, z. B. Speicher- oder Festplattenplatz, schwer zu verwalten.

[0004] Weiterhin unterscheidet sich Softwaremiete bedeutend von einem Abonnement eines Netzdienstes. Man nehme beispielsweise an, daß ein Softwareverkäufer einen Server bereitstellt, an den sich Kunden über ihre Softwareclienten anschalten. Während der Verbindungszeit überprüft der Server Nutzungsaufzeichnungen, z. B. Verbindungszeit. Der Verkäufer bewertet Gebühren auf Grundlage der auf dem Auditweg des Servers aufgezeichneten Informationen. Dieses Client/Server-Beispiel unterscheidet sich von der vorliegenden Erfindung, da eine Gegenwart des Softwareverkäufers online nicht unbedingt erforderlich ist. Stattdessen wird vom Kunden die Software ohne jegliche erforderliche Netzverbindungen ablaufen gelassen, nachdem er die Erlaubnis zur Verwendung der gemieteten Software erhalten hat. Weiterhin hindert der Bestellungsdienst den Kunden nicht daran, häufig benutzte Posten im Cache abzuspeichern.

[0005] Aus [2] ist eine Übersicht asymmetrischer Kryptographie, beispielsweise des RSA-Schemas, und probabilistischer Verschlüsselung, beispielsweise des Blum-Goldwasser-Schemas probabilistischer Verschlüsselungen mit öffentlichen Schlüsselwörtern ersichtlich.

[0006] Die internationale Patentanmeldung WO 88 05941 lehrt ein Software-Regelungssystem zum Regeln der Verwendung eines Softwareprogramms in einem Host-Digitaldatenverarbeitungssystem. Das Softwareregelungssystem enthält eine oder mehrere, durch das Softwareprogramm verarbeitete Fixpunktroutinen und eine Softwareregelungsvorrichtung, die Teil des Computersystems oder extern daran angeschlossen sein kann. Die Fixpunktroutinen erzeugen zufallsmäßige Fixpunktnachrichten, die verschlüsselt und zur Softwareregelungsvorrichtung übertragen werden. Die Softwareregelungsvorrichtung entschlüsselt die Fixpunktnachricht, führt eine Verarbeitungsoperation zum Erzeugen einer Antwortnachricht durch, verschlüsselt die Antwort und sendet die verschlüsselte Antwort zur Fixpunktroutine. Die Fixpunktroutine bestimmt dann, ob die verschlüsselte Antwort richtig ist und erlaubt entweder, daß das Softwareprogramm weitermacht, oder beendet es.

[0007] Aus [2] ist eine Übersicht verschiedener probabilistischer Nachweisverfahren, beispielsweise Nachweisverfahren ohne Kenntnis (z. B. Feige-Fi-

at-Shamir-Verfahren, Guillou-Quisquater-Verfahren, Blum-Feldmann-Micali-Verfahren, Brassard-Verfahren, Crepau-Verfahren usw.) oder zeugenverbergende Nachweisverfahren (z. B. Feige-Shamir-Verfahren usw.) ersichtlich.

[0008] Aus [2] ist eine Übersicht von Verfahren digitaler Signatur (z. B. Rivest-Shamir-Adleman usw.) und eine formelle mathematische Definition von digitalen Signaturen ersichtlich.

[0009] MD5 [3] ist ein Beispiel einer (ansonsten als Einweg-Hash-Funktion bekannten) Message-Digest-Funktion. Es ist rechnerisch unausführbar oder sehr schwierig, den Kehrwert einer Message-Digest-Funktion zu berechnen.

[0010] In [4] wird die aus Luftturbulenz in Festplattenlaufwerken entstehende kryptographische Zufallsmäßigkeit beschrieben.

[0011] In [5] werden die Chi-Quadrat-Prüfung, die Kolmogorov-Smirnov-Prüfung und die Prüfung der seriellen Korrelation beschrieben.

[0012] Ein asymmetrischer Verschlüsselungsmechanismus enthält öffentliches Verschlüsselungsmaterial und entsprechendes privates Verschlüsselungsmaterial. Es ist rechnerisch unausführbar, das private Verschlüsselungsmaterial zu berechnen, wenn keine weiteren Informationen als das entsprechende öffentliche Verschlüsselungsmaterial zur Verfügung stehen. In der vorliegenden Erfindung wird asymmetrische Kryptographie in Wechselwirkungen zwischen zwei Teilnehmern A und B benutzt. A beweist B, daß A Zugang zu privatem Verschlüsselungsmaterial hat und B validiert den Beweis. A offenbart B nicht das private Verschlüsselungsmaterial.

Verfahren der digitalen Signatur

[0013] Eine digitale Signatur ist ein elektronisches Analog einer handschriftlichen Unterschrift. An einem Nachweis einer digitalen Signatur sind mindestens zwei Teilnehmer A und B beteiligt. Nach Aufgabe seines oder ihres öffentlichen Verschlüsselungsmaterials an eine öffentliche Stelle wird von A eine Nachricht unter Verwendung des privaten Verschlüsselungsmaterials verschlüsselt. Da jeder auf das öffentliche Verschlüsselungsmaterial zugreifen kann, gibt es keine Nachrichtengeheimhaltung. Da jedoch A der einzige Kunde mit Zugang zu dem privaten Verschlüsselungsmaterial ist, kann sonst niemand die Signatur von A durch Durchführung der Verschlüsselung fälschen. Die Signatur von A kann von jedem unter Verwendung des öffentlichen Verschlüsselungsmaterials validiert werden, indem sie einfach unter Verwendung des öffentlichen Verschlüsselungsmaterials von A entschlüsselt wird.

[0014] An einem asymmetrischen Vertraulichkeitsnachweis sind mindestens zwei Teilnehmer A und B beteiligt. A besitzt privates Verschlüsselungsmaterial und B hat keinen Zugriff auf das private Verschlüsselungsmaterial von A, es sei denn B offenbart selbst das private Verschlüsselungsmaterial (was B nicht tun sollte). Zu Beginn besitzen A und B kein gemeinsames Geheimnis. Während des Verfahrens erfahren A und B ein gemeinsames Geheimnis.

[0015] In allen asymmetrischen Verschlüsselungsverfahren kann jeder Kunde sein oder ihr öffentliches Verschlüsselungsmaterial an ein Verzeichnis mit öffentlichem Zugang aufgeben, ohne das entsprechende private Verschlüsselungsmaterial zu kompromittieren. Der Kunde sollte gewöhnlich sein oder ihr privates Verschlüsselungsmaterial als strenges Geheimnis schützen; sonst kann das Verschlüsselungssystem nicht die Richtigkeit (Geheimhaltung) garantieren. Der am besten bekannte Mechanismus zum Schützen seines privaten Verschlüsselungsmaterials besteht in der Verwendung einer Chipkarte. Im vorliegenden Fall ist die Chipkarte eine Vorrichtung ohne Schnittstelle zum Freigeben von privatem Verschlüsselungsmaterial (in nicht kryptographisch geschützter Form). Alle Verschlüsselungsoperationen, die direkt auf das private Verschlüsselungsmaterial Bezug nehmen, werden auf der Chipkarte selbst durchgeführt. Infolgedessen kann niemand den Inhalt des auf einer Chipkarte gespeicherten privaten Verschlüsselungsmaterials entdecken.

[0016] Obwohl Chipkarten den besten Schutz bieten, können soziale Faktoren des elektronischen Handels eine Rolle bei der Sicherstellung von Schutz für privates Verschlüsselungsmaterial bieten. Eine der wesentlichen Schwierigkeiten, die mit asymmetrischen Verschlüsselungsdiensten verbunden sind, ist die Authentifizierung. Wenn beispielsweise A sein oder ihr öffentliches Verschlüsselungsmaterial an ein öffentliches Verzeichnis aufgibt, wie kann dann B die Gültigkeit bewerten? Das heißt ein Pirat kann versuchen, sich als A auszugeben, aber das Schlüsselmaterial des Piraten aufgeben. Von einigen Handelsorganisationen werden Lösungen für dieses Problem geboten, indem sie als Zertifizierungsstelle (CA – Certification Authority) wirken. Gegen (möglicherweise) eine Gebühr erbittet die CA von möglichen Kunden Identifikationsmaterial wie beispielsweise einen Führerschein oder einen Paß. Nach Validierung des Identifizierungsmaterials gibt die CA das öffentliche Verschlüsselungsmaterial des Kundens an ein öffentliche Verzeichnis auf und die CA unterzeichnet (unter Verwendung einer digitalen Signatur mit dem privaten Schlüssel der CA) ein Zertifikat, in dem das öffentliche Verschlüsselungsmaterial des Kundens abgespeichert ist. Um die Verwendung von Verzeichnissen, die öffentliches Verschlüsselungsmaterial enthalten, zu erleichtern, können standardisierte Dienste, wie beispielsweise X.500 benutzt werden.

[0017] Nachdem ein Kunde sein oder ihr öffentliches Verschlüsselungsmaterial an die CA aufgegeben hat, wird er oder sie sich wahrscheinlich sehr bemühen, sein oder ihr privates Verschlüsselungsmaterial zu schützen. Bei manchen asymmetrischen Schlüsseln würde, wenn das private Verschlüsselungsmaterial des Kundens unwissentlich kompromittiert werden würde, der Kunde Grund für bedeu-

tende Sorge haben. Beispielsweise könnten im Fall von RSA-Schlüsseln, die auch für digitale Signaturen benutzt werden können, vernetzte Verkäufer möglicherweise elektronische Handelstransaktionen autorisieren.

[0018] Es ist die Aufgabe der vorliegenden Erfindung, ein kryptographisch sicheres Softwaremietsystem zu erstellen.

Kurzbeschreibung der Erfindung

[0019] Die Erfindung wird durch die Merkmale der beiliegenden unabhängigen Ansprüche 1, 26 und 27 definiert. Weitere Aspekte der Erfindung werden durch die Merkmale der abhängigen Ansprüche 2 bis 25 und 28 bis 31 definiert.

[0020] Es wird ein neuartiger Mechanismus bereitgestellt, bei dem die Anzahl erforderlicher SUSL keine Funktion der Anzahl gemieteter Programme ist. Insbesondere erfordert die vorliegende Erfindung nur eine einzige SUSL ungeachtet der Anzahl gemieteter Programme oder ob diese Programme gleichzeitig ablaufen oder nicht. Beispielsweise könnte ein Kunde möglicherweise gleichzeitig 100 oder mehr gemietete Programme ablaufen lassen und dabei nur eine einzige SUSL benutzen. Die SUSL benötigt zum Schützen einer unbegrenzten Anzahl von gemieteten Programmen nicht mehr als 128 Bit. In manchen Fällen kann eine einzige 32-Bit-SUSL oder eine einzige 64-Bit-SUSL genügen. Wie später beschrieben wird, wird die SUSL zur Bezeichnung einer einzigen nicht negativen Ganzzahl benutzt. Diese Ganzzahl wird als Zähler benutzt, der mit einem Anfangswert (normalerweise Null) beginnt und mit dem höchsten Wert endet, der durch den Zähler dargestellt werden kann. Unter normalen Umständen ist es nicht möglich, alle möglichen Nummern durchzuzählen, die durch einen 128-Bit-Zähler dargestellt werden können. Selbst wenn man eine Maschine bauen würde, die den Zähler Trillionen von Malen pro Sekunde erhöhen könnte (man würde normalerweise nicht so schnell zählen wollen), würde es Millionen von Jahren dauern, den Höchstwert des Zählers ($2^{128} - 1$) zu erreichen.

[0021] Auch bietet die vorliegende Erfindung ein neues Verfahren zum Verteilen gemieteter Software. Bei diesem Verfahren kann ein Kunde von jedem beliebigen Verkäufer neue Anwendungen jedes Mal, wenn er oder sie es wünscht, mieten. Der Kunde muß nicht neue externe Mietvorrichtungen oder neue SUSL installieren.

[0022] Vom Gesichtspunkt der Sicherheit besteht die Hauptmotivierung dafür, mindestens eine SUSL zu erfordern, darin, vor der Attacke der Sicherung und Wiederherstellung, wie unten beschrieben, zu schützen. Man nehme an, daß eine gemietete Anwendung keine SUSL erfordert. Ein Angreifer kann durch Durchführung der folgenden Schritte die Sicherheit durchbrechen. Als erstes mietet der Angreifer in der Rolle eines berechtigten Mietkunden ganz legal ein Programm. Als nächstes baut sich der Angreifer eine volle Systemsicherungskopie des gesamten nichtflüchtigen Speichers auf der Maschine des Angreifers. Als nächstes führt der Angreifer das gemietete Programm aus. Als nächstes schaltet der Angreifer ab und startet seine oder ihre Maschine von Neuem. Abschließend stellt der Angreifer wieder den Zustand des nichtflüchtigen Speichers zum Ursprungszustand zur Zeit der Sicherung her. An dieser Stelle hat der Angreifer jede mögliche Aufzeichnung zerstört, daß das Programm vorher gemietet war. Mit einer SUSL wird andererseits diesem Angriff entgegengewirkt, da eine SUSL die Eigenschaft aufweist, daß die in der SUSL enthaltenen Informationen nicht auf unberechtigte Weise abgeändert werden können. Infolgedessen kann die SUSL nicht durch die Wiederherstellungsoperation des Angreifers mit dem Zweck des Durchbrechens der Sicherheit überschrieben werden.

[0023] Da nicht sichergestellt werden kann, daß ein Kunde zum Schluß der Mietzeit alle Kopien der Software löscht, wird ein alternativer Ansatz bereitgestellt. Die Software wird so verriegelt, daß der Kunde die Software nicht ausführen kann, ohne daß er zuerst einen entsprechenden Schlüssel bereitstellt. Am Ende der Mietzeit wird dem Kunden verboten, diesen Schlüssel oder irgendeine Kopie des Schlüssels nachfolgend zu benutzen. Zum Schluß der Mietzeit kann der Kunde daher die Software nicht länger benutzen, da er nicht länger in der Lage sein wird, einen entsprechenden Schlüssel bereitzustellen.

Kurze Beschreibung der Zeichnungen

[0024] **Fig.** 1 ist ein Blockschaltbild der Architektur des Software-Mietsystems.

[0025] **Fig.** 2 ist ein Blockschaltbild der Softwarekomponenten, die auf der Maschine des Kunden installiert werden müssen.

[0026] **Fig.** 3 ist ein Flußdiagramm der Funktionsweise eines zur Erzeugung von Augenblickswerten benutzten Zufallszahlengenerators.

[0027] **Fig.** 4 ist ein Blockschaltbild eines Kaufszenarios von Mietsoftware.

[0028] **Fig.** 5 ist ein Blockschaltbild der Chipkartenarchitektur.

[0029] **Fig.** 6 ist ein Blockschaltbild eines beispielhaften Auditweges.

Beschreibung einer Ausführungsform der Erfindung

[0030] Es wird nunmehr als Beispiel eine Ausführungsform entsprechend der Erfindung unter Bezugnahme auf die beiliegenden Zeichnungen beschrieben.

[0031] **Fig.** 1 zeigt die Architektur des Systems **100**. Auf dem System sitzen möglicherweise mehrere Anwendungen (Programme) von Software, wobei jede Anwendung ihre eigene Schlüsseldatei aufweist, was ausführlich später beschrieben wird. Die **Fig.** 1 zeigt drei Anwendungen, ein Textverarbeitungsprogramm

104, ein Tabellenkalkulationsprogramm **105** und eine andere Anwendung **106**, die auf Schlüsseldateien **101**, **102** bzw. **103** zugreifen. In manchen Fällen können sich mehrere Anwendungen **104**, **105**, **106** eine gemeinsame Schlüsseldatei teilen.

[0032] Jede der Anwendungen **104**, **105**, **106** greift auf ihre Schlüsseldatei **101**, **102** und **103** zu, um das öffentliche Verschlüsselungsmaterial des Kundens, wie später ausführlich beschrieben, zu entnehmen.

[0033] Jeder Verkäufer einer Anwendung setzt Mietanweisungen in ein kopiergeschütztes Programm ein. Diese Mietanweisungen erstellen Protokollaufzeichnungen, z. B. **109**, **110**, **111**. Beispielsweise erstellt das Textverarbeitungsprogramm **104** alle fünfzehn Minuten während seines Ablaufs die folgende Protokollaufzeichnung: „Word Process WP with public key 9828a8c12a5873654bac684517d3afe3 executed for 15 minutes" [Textprogramm WP mit öffentlichem Schlüssel 9828a8c12a5873654bac684517d3afe3 15 Minuten lang ausgeführt] (man beachte, daß in der Aufzeichnung die Message-Digest-Funktion des öffentlichen Verschlüsselungsmaterials anstelle des öffentlichen Verschlüsselungsmaterials selbst abgespeichert sein könnte). Als nächstes sendet die Anwendung ihre Protokollaufzeichnung zu einem Mietserver **107**. Der Mietserver **107** setzt die Protokollaufzeichnung am Ende eines sicheren Auditweges **108** ein, der an einer möglicherweise ungesicherten Speicherstelle, z. B. einer Datei auf einer Platte, abgespeichert ist. Zur Sicherung verläßt sich der Mietserver **107** auf die Unterstützung einer Chipkarte **112**.

[0034] Eine Anwendung, z. B. **104**, **105** oder **106**, wünscht beispielsweise, eine Protokollaufzeichnung zu erstellen, die eine beliebige willkürliche Kette von Bit mit willkürlicher Länge enthält. Zusätzlich oder anstelle der Aufzeichnung von Zeit könnte eine Anwendung möglicherweise protokollieren, wieviel Male die Anwendung oder einige ihrer Modulen ablaufen. Beispielsweise kann der SS **105** möglicherweise bei jeder Initialisierung von SS eine einzelne Protokollaufzeichnung „Anwendung SS mit öffentlichem Schlüssel 768230aac8239d9df88cfe3c7b832a läuft ab" anhängen. Auf demselben Auditweg können verschiedene Arten von Auditaufzeichnungen, z. B. Benutzungszeit oder wieviele Male die Benutzung stattfand, erscheinen. Mehrere gemietete Anwendungen können gleichzeitig denselben Auditweg benutzen.

[0035] Softwaremiete wird durch Vergleichen von Schwellwerten mit dem Auditweg erhalten. In **Fig.** 4 ist ein Kunde **402** dargestellt, der Software von einem Verkäufer **401** mietet. Zuerst sendet der Kunde **402** eine Anforderung zum Mieten der Software an den Verkäufer **401** in einer Auftragsanforderung **403**. Im vorliegenden Beispiel kauft der Kunde sechs Stunden der Anwendung **104**. Nach Empfang der Bezahlung sendet der Verkäufer dem Kunden eine Schlüsseldatei **404**, die eine Benutzungsberechtigung enthält. Im vorliegenden Fall erlaubt die Schlüsseldatei **404** sechs Stunden Ausführung der Anwendung **104**. Die später beschriebene Schlüsseldatei **404** kann möglicherweise andere Informationen, z. B. Kopierschutz- oder Lizenzierungsinformationen enthalten.

[0036] Der Auditweg **108** wird periodisch von der gemieteten Anwendung, z. B. Textverarbeitungsprogramm (Anwendung WP) **104** untersucht. Wenn der Auditweg **108** nicht gültig ist, dann erlaubt das Textverarbeitungsprogramm **104** nicht, gemietet zu werden. Wenn jedoch der Auditweg **108** gültig ist, dann wertet die Anwendung **104** den Auditweg aus und vergleicht die Auswertung mit der Schlüsseldatei **404**. Beispielsweise zählt die Anwendung **104** die Anzahl von Protokollaufzeichnungen, die 15-Minuten-Zeitabstände beschreiben. Als nächstes schlägt die Anwendung **104** in der Schlüsseldatei **404** nach, um einen Mietschwellwert zu finden, der im vorliegenden Beispiel 6 Stunden beträgt (24 × 15-Minuten-Zeitabstände). Wenn die Anwendung **104** weniger als 24 ihrer Protokollaufzeichnungen mit 15-Minuten-Zeitabständen findet, dann läuft die Anwendung **104** weiterhin ab. Ansonsten erlaubt die Anwendung **104** nicht, gemietet zu werden. Im letzteren Fall muß der Kunde eine neue Schlüsseldatei kaufen, um die Anwendung **104** weiter zu mieten. Sollte die Anwendung **104** ihren Mietschwellwert überschreiten, dann würden andere Anwendungen, z. B. Tabellenkalkulation **105** und die andere Anwendung **106** nicht bewirkt werden. Das heißt jede gemietete Anwendung betrachtet ihre eigenen Aufzeichnungen vom Auditweg ohne Auslegung von durch andere Anwendungen erstellten Aufzeichnungen.

[0037] Aus der obigen Besprechung ist ersichtlich, daß die Architektur Softwaremiete realisiert, vorausgesetzt, daß die gemieteten Anwendungen, z. B. **104**, **105**, **106**, den Auditweg **108** unzweideutig validieren können. Es sollten folgende Eigenschaften erfüllt sein:

1. Löcher: eine gemietete Anwendung, z. B. das Textverarbeitungsprogramm **104** im vorliegenden Beispiel validiert, daß der Auditweg alle Aufzeichnungen enthält, die jemals geschrieben worden sind, ungeachtet der Anwendung. Wenn eine Anwendung vorher zehn Protokollaufzeichnungen geschrieben hat, dann würde die gemietete Anwendung, z. B. das Textverarbeitungsprogramm **104**, nicht den Auditweg validieren, wenn die gemietete Anwendung nicht alle zehn Protokollaufzeichnungen finden könnte. Erforderlich ist eine Abwesenheit von Löchern, da nicht gewünscht wird, einem Angreifer zu erlauben, einzelne Protokollaufzeichnungen zu löschen, um eine Benutzungsaufzeichnung zu zerstören.
2. Abänderung: Eine Anwendung, z. B. das Textverarbeitungsprogramm **104**, muß unzweideutig zu dem Schluß kommen, daß keine der Protokollaufzeichnungen des WP durch irgendeinen unberechtigten Angreifer abgeändert worden sind. Ansonsten könnte beispielsweise der Angreifer alle 15-Minuten-Protokollaufzeichnungen in 15-Sekunden-Protokollaufzeichnungen abändern, um

die Zeitdauer drastisch zu erhöhen, für die die Software ablaufen kann.

3. Aktuell: Eine gemietete Anwendung muß in der Lage sein, zu validieren, daß der Auditweg **108** aktuell ist. Ansonsten könnte der Auditweg **108** möglicherweise alt sein und dadurch relativ neue Auditaufzeichnungen **109**, **110**, **111** verbergen. Man würde beispielsweise nicht wünschen, daß ein Angreifer den Sicherungs- und Wiederherstellungsangriff durchführt.

[0038] Mit diesen drei Eigenschaften wird jeder Anreiz für einen Angreifer eliminiert, einen Auditweg **108** zu verfälschen, zu löschen, zu verlieren oder sonst zu mißbrauchen. Würde der Angreifer den Auditweg **108** ungültig machen, dann würden alle gemieteten Anwendungen **104**, **105**, **106** den Mißbrauch erkennen und danach die Miete verweigern.

[0039] Um Sicherheit bereitzustellen erfordert die Architektur eine Chipkarte **112**, die asymmetrische Kryptographie durchführt. Im vorliegenden Beispiel führt die Chipkarte **112** digitale Signaturen aus. Die Chipkarte **112** enthält privates Verschlüsselungsmaterial **501** und einen Zähler **502**, (siehe **Fig.** 5).

[0040] Wenn ein Kunde die Chipkarte **112** erhält, befindet sich die Chipkarte **112** in einem sicheren Zustand. Die Chipkarte bietet genau zwei Dienste, mit denen entweder auf das private Verschlüsselungsmaterial oder den Zähler zugegriffen werden kann: SignAndIncrement() und GetCounter(), die hiernach in Pseudo-Programmcode beschrieben werden (man beachte, daß das Symbol // einen Kommentar bezeichnet und ← den Zuweisungsoperator bezeichnet):

```
Signature SignAndIncrement (HASH h)// h ist eine Message-Digest-Funktion
BEGIN
[1] Berechnen der Message-Digest-Funktion von h und des Chipkartenzählers, d. h. h' ← hash(h,counter)
[2] Signieren h' mit dem privaten Verschlüsselungsmaterial
[3] Erhöhen des Chipkartenzählers um 1
[4] Zurücksenden der im Schritt [2] berechneten digitalen Signatur
END
integer GetCounter()
BEGIN
[1] Zurücksenden des aktuellen Werts des Chipkartenzählers
END
```

[0041] Man betrachte die folgende beispielhafte Spur. Angenommen, der Chipkartenzähler weist einen Anfangswert von 6 auf und daß folgende Operationen ausgeführt werden:

(i) Signature1 ← SignAndIncrement (hash(„m1"))
(ii) Signature2 ← SignAndIncrement (hash(„m2"))
(iii) int1 ← GetCounter()

[0042] Die Ergebnisse dieses Beispiels sind:

– Signature1 erlangt die digitale Signatur (unter Verwendung des privaten Verschlüsselungsmaterials der Chipkarte) von hash (hash(„m1"),6)
– Signature2 erlangt die digitale Signatur (unter Verwendung des privaten Verschlüsselungsmaterials der Chipkarte) von hash (hash(„m2"),7)
– int1 erlangt 8

[0043] Der Auditweg **108** enthält eine Liste von Aufzeichnungen, wobei jede Aufzeichnung vier Felder aufweist: Nonce (Augenblickszahl), String (Kette), Counter (Zähler) und Signature (Signatur). Die Dateneingabe in die Signatur ist hash(hash(nonce,string),counter). **Fig.** 6 zeigt einen beispielhaften Auditweg mit vier Aufzeichnungen. In der ersten Aufzeichnung weist die Augenblickszahl den Wert 96 auf, die Kette ist „WP 15 minutes public key 9828a8c12a5873654bac684517d3afe3" (wobei 9828a8c12a5873654bac684517d3afe3 die Message-Digest-Funktion eines wirklichen öffentlichen Schlüssels bezeichnet), der Zählerwert beträgt 0 und die digitale Signatur ist hash(hash(96,"WP 15 minutes public key 9828a8c12a5873654bac684517d3afe3"),0). Hier wurde die digitale Signatur unter Verwendung des privaten Verschlüsselungsmaterials der Chipkarte bereitgestellt, das dem öffentlichen Verschlüsselungsmaterial 9828a8c12a5873654bac684517d3afe3 entspricht. Dieses öffentliche Verschlüsselungsmaterial kann aus der Schlüsseldatei von WP entnommen werden.

[0044] Der Zähler wird niemals von seinem Höchstwert auf Null zurückgestellt. Wenn der Zähler seinen Höchstwert, z. B. 2^{128} – 1 erreicht, hält das System an.

[0045] Eine gemietete Anwendung hängt eine Aufzeichnung an den Auditweg an, indem sie die Write-Routine ausführt, wobei die Write-Routine in die gemieteten Anwendungen eingebettet ist. Diese Routine erzeugt eine Auditaufzeichnung und sendet dann die Auditaufzeichnung zum Mietserver **107**. Der Mietserver **107** schreibt die Auditaufzeichnung in eine nichtflüchtige stabile Abbildung des Auditwegs, z. B. auf einer oder mehreren Dateien. Der Mietserver **107** synchronisiert den Zugriff auf die Chipkarte **112** und den Auditweg. Der Mietserver **107** kann nicht so ablaufen, daß die Systemsicherheit vereitelt wird, d. h. die gemieteten Anwendungen mißtrauen dem Mietserver **107**. Sollte der Mietserver **107** nicht ordnungsgemäß handeln, könnte möglicherweise eine Dienstverweigerung stattfinden, da es möglich sein könnte, daß die gemieteten Anwendungen den Auditweg nicht validieren können.

[0046] Die Write-Routine wird unten in Pseudo-Programmcode geboten:

```
Boolesch Write(String str)
BEGIN
[1] n ← generate a new nonce
[2] h1 ← hash(n,str)
[3] s ← SignAndIncrement (h1)
// unten ist c eine lokale Kopie des Werts in der Chip-
```

karte
[4] c ← GetCounter()
// unten hat Erniedrigung um 1 eine Auswirkung auf die Chipkarte
[5] decrement c by 1
[6] h2 ← hash(h1,c)
[7] Validieren, daß s die Signatur von h2 ist, gegen den in der Schlüsseldatei aufgefundenen öffentlichen Schlüssel (wenn die Validierung fehlschlägt, dann sofort Ausfall zurücksenden, ohne weitere Schritte auszuführen; die Schlüsseldatei wird später ausführlich beschrieben).
[8] Auditweg r ← <n,str,c,s> erstellen
[9] Anhängen r an Auditweg
[10] Zurücksenden TRUE, wenn alle vorhergehenden Schritte erfolgreich sind, sonst Ausfall zurücksenden
END

[0047] Die ValidateTrail-Routine ist ebenfalls in die gemietete Anwendung eingebettet und sollte periodisch ausgeführt werden und wird unten in Pseudo-Programmcode geboten (es sei angenommen, das System begann mit einem Zähler-Anfangswert von Null):

Boolesch ValidateTrail()
BEGIN
[1] c ← GetCounter()
[2] Write (c) // obige Write-Routine benutzen, Austre
// ten, wenn Fehler
[3] r ← Last record in the audit trail // Dies ist die eben im Schritt [2] geschriebene Aufzeichnung
[4] Validieren der in r gespeicherten Signatur gegen den in der Schlüsseldatei gespeicherten öffentlichen Schlüssel
[5] Validieren, daß c das gleiche wie der in r gespeicherte Zähler ist
[6] FOR i ← 0, i um 1 ERHÖHEN, BIS keine Aufzeichnungen mehr da sind
[6.1] r ← I-te Aufzeichnung vom Auditweg
[6.2] Validieren der in r gespeicherten Signatur gegen den in der Schlüsseldatei gespeicherten öffentlichen Schlüssel
// Signaturvalidierung umfaßt die Nachricht
// Digest-Neuberechnung
[6.3] Validieren, daß i das gleiche wie der in r gespeicherte Zähler ist
[7] END FOR LOOP
[8] wenn alle obigen Schritte erfolgreich sind, dann TRUE zurücksenden, ansonsten FALSE zurücksenden
END

[0048] In Schritten [4] und [6.2] sind alle Eingaben für die Validierung von der Auditaufzeichnung selbst. Durch sorgfältige Auswertung aller Schritte in den Routinen Write und ValidateTrail ist klar, daß jeder Angriff, der die beabsichtigte Verwendung dieser Routinen vereitelt, einen Ausfall verursacht. In diesem Fall wird der Ausfall von den gemieteten Anwendungen erkannt, und sie gestatten nicht, gemietet zu werden.

[0049] Der Mechanismus zur Wiederherstellung nach einem Ausfall ist von dem Verkäufer für die jeweilige gemietete Anwendung abhängig. Der Verkäufer kann beispielsweise auf Anforderung irgendeines Kundens eine neue Schlüsseldatei ausgeben. Die bedeutendste Frage ist vielleicht die Wiederherstellung nach einem unbeabsichtigten Verlust des Auditweges, der aufgrund eines Plattenfehlers auftreten könnte. Um Schutz gegen diese Situation zu bieten, sollte der Mietserver **107** für das Aufschreiben aller Aufzeichnungen auf dem Auditweg für alle gemieteten Anwendungen verantwortlich sein. Der Mietserver sollte auf der lokalen Festplatte einen Hauptauditweg und auf einem getrennten Medium, z. B. einer Diskette oder einem Netz-Dateiserver mindestens einen Reserve-Auditweg unterhalten. Man kann beispielsweise einen Dienst bereitstellen, der dem Mietserver erlaubt, Auditwege per Email zu einem gesicherten Netz-Sicherungsdienst zu senden. In diesem Fall kann Geheimhaltung sichergestellt werden, indem dem Mietserver erlaubt wird, den Auditweg vor Übertragung zu verschlüsseln.

Softwarekomponenten (siehe **Fig.** 2)

[0050] Der Begriff Abfragemittel (Challenge Means) wird dazu benutzt, den Teil eines gemieteten Programms zu bezeichnen, der Komponenten der vorliegenden Erfindung implementiert. Das Abfragemittel umfaßt die Implementierung der Routinen Write und ValidateTrail. Zusätzlich weist das Abfragemittel den Rest des Programms in Abhängigkeit von den Ergebnissen der Routinen Write und ValidateTrail dahingehend an, wie er ablaufen soll. Zusätzlich steht das Abfragemittel wie unten beschrieben mit der Schlüsseldatei des Programms in Wechselwirkung.

Schlüsseldatei **404**

[0051] Die Erstellung der Schlüsseldatei **404** wird durch einen Schlüsseldateigenerator durchgeführt, der ein Programm ist, das auf der Einrichtung des Verkäufers abläuft. Der Verkäufer **401** muß dafür sorgen, dieses Programm zu beschützen.

[0052] Bei der Verwendung des Schlüsseldateigenerators gibt ein Bediener folgende Informationen ein:

[0053] Verkäufername: Verkäufername ist der Name der Firma des Verkäufers.

[0054] Verkäuferpaßwort: Das Verkäuferpaßwort ist das Paßwort, das das private Verschlüsselungsmaterial der Verkäuferfirma freigibt. Firmenmitarbeiter, die das Paßwort nicht kennen, können keine Schlüsseldateien erstellen.

[0055] Kundenname: Der Kundenname ist der ausgeprägte Name eines Kundens (in [2] definiert), für den eine Schlüsseldatei zu erstellen ist. Der Name wird in eine Datenbank mit öffentlichem Verschlüsselungsmaterial indexiert.

[0056] Schlüsseldateiname: Der Schlüsseldateina-

me ist der Name einer neuen Schlüsseldatei.

[0057] Nach Erlangen dieser Informationen baut der Schlüsseldateigenerator die Schlüsseldatei **404** auf, die eine später beschriebene Kundeninformationskette (CIS – Customer Information String) enthält. Teile der Schlüsseldatei **404** erscheinen dem Kunden **402** als eine vollständig zufallsmäßige Folge von Werten.

[0058] Zum Aufbauen der Schlüsseldatei **404** gehören folgende Operationen.

[0059] Als erstes erstellt der Schlüsseldateigenerator eine Datei und setzt das öffentliche Verschlüsselungsmaterial des Kundens zusammen mit tausenden Täuschbit in die Datei ein. Im vorliegenden Beispiel enthält jede Schlüsseldatei **404** annähernd 480000 Täuschbit. Diese Anzahl Bit stellt eine bedeutende Menge Täuschmaterial dar und kann dennoch in eine standardmäßige Email-Nachricht passen.

[0060] Jede Schlüsseldatei **404** speichert die CIS an einer anderen Stelle. Zusätzlich sind in jeder Schlüsseldatei **404** verschlüsselte Kundeninformationen eingebettet, ohne den erforderlichen Verschlüsselungsschlüssel zu offenbaren. Diese verschlüsselten Kundeninformationen erlauben einem Verkäufer, den Eigentümer einer Schlüsseldatei **404** leicht zu identifizieren, sollte die Schlüsseldatei **404** an einer öffentlichen Stelle wie beispielsweise einem Mitteilungsbrett erscheinen. Vom Schlüsseldateigenerator wird dann die Schlüsseldatei (oder Teile der Schlüsseldatei) **705** mehrere Male unter Verwendung unterschiedlicher Algorithmen verschlüsselt und wieder verschlüsselt. Abschließend signiert der Schlüsseldateigenerator die Schlüsseldatei **404** unter Verwendung des privaten Verschlüsselungsmaterials des Verkäufers durch Anwenden eines Algorithmus für eine digitale Signatur.

[0061] Eine Schlüsseldatei wird als validiert bezeichnet, wenn das Abfragemittel der gemieteten Anwendung (unten beschrieben) die Signatur des Verkäufers unter Verwendung des im Binärcode des Abfragemittels gespeicherten öffentlichen Verschlüsselungsmaterials validieren und auf die in der Schlüsseldatei **404** gespeicherte entschlüsselte CIS zugreifen kann.

[0062] Nach Erstellung der Schlüsseldatei **404** sendet der Rechner des Verkäufers die Schlüsseldatei **404** per Email zum Kunden **402**.

[0063] Die CIS ist eine Kette, die das öffentliche Verschlüsselungsmaterial des Kundens, die Zugriffsrechte des Kundens und möglicherweise einen Mietschwellwert enthält. Die Zugriffsrechte liefern Informationen zum gezielten Freigeben von Funktionalität. Beispielsweise kann ein Zugriffsrecht möglicherweise das Programm zum Ausführen der Druckfunktion berechtigen. Ein anderes Zugriffsrecht konnte möglicherweise der Anwendung erlauben, Toninformationen zu den Lautsprechern des Kundens zu senden. Es ist anzunehmen, daß hochvertrauenswürdige Kunden oder Kunden, die mehr Geld bezahlen, bessere Zugriffsrechte in ihren Schlüsseldateien erhalten. Die Schlüsseldatei **404** kann mehrere Verwendungen aufweisen, z. B. Berechtigung für die Anwendung und Mietinformationen.

[0064] Im folgenden wird die Funktionsweise des Mietmechanismus ausführlicher beschrieben. Dies wird bei dem erstmaligen Versuch des Kundens, das gemietete Programm auszuführen, durchgeführt.

[0065] Wenn der Abfragemechanismus **202** den Vorgang beginnt, greift der Abfragemechanismus **202** (siehe **Fig.** 7) auf die mit der gemieteten Anwendung verbundene Schlüsseldatei **404** zu und ruft eine Signaturvalidierungsfunktion **203** im Abfragemechanismus **202** auf, um unter Verwendung des öffentlichen Verschlüsselungsmaterials **201** des Verkäufers, das im Abfragemechanismus **202** eingebettet ist, die Verkäufersignatur der Schlüsseldatei **404** zu validieren. Durch diese Validierung der Schlüsseldateisignatur wird sichergestellt, daß ein Angreifer die Schlüsseldatei **404** oder ihre digitale Signatur nicht ohne zusätzliche Abänderung des Abfragemechanismus **202** abändern kann. Der Verkäufer **401** kann diesen Schutz wahlweise unter Verwendung zusätzlicher herstellerspezifischer Verteidigungsmittel verstärken. Wenn die Schlüsseldatei **404** abgeändert worden ist, blockiert der Abfragemechanismus **202** das gemietete Programm oder stört auf andere Weise die normale Programmausführung.

[0066] Angenommen die Signatur der Schlüsseldatei **404** ist validiert, dann wird die Schlüsseldatei **404** unter Verwendung eines herstellerspezifischen, verkäuferspezifischen Algorithmus vom Abfragemechanismus **202** syntaxanalytisch zerlegt, um das öffentliche Verschlüsselungsmaterial des Kundens in der Schlüsseldatei **404** zu finden, und das öffentliche Verschlüsselungsmaterial des Kundens entnommen.

[0067] Danach wird, wie schon beschrieben, das Software-Mietprotokoll jedes Mal, wenn es vom Software-Mietmechanismus erfordert wird, vom Abfragemechanismus (Abfragemittel) ausgeführt. Das heißt, das Abfragemittel hängt, wie oben beschrieben, Auditaufzeichnungen sicher an den Auditweg an.

Nonce-Generator

[0068] Die Erzeugung einer Augenblickszahl (Nonce) wird durch einen im Abfragemechanismus **202** enthaltenen Nonce-Generator durchgeführt. Die Funktionsweise des Nonce-Generators ist wie folgt (siehe **Fig.** 3).

[0069] Als erstes fragt der Nonce-Generator eine große Anzahl von Systemparametern, z. B. die Systemzeit, den in der Seitentabelle freibleibenden Raum, die Anzahl logischer Plattenlaufwerke, die Namen der Dateien im Verzeichnis des Betriebssystems usw. ab.

[0070] Als nächstes wird vom Nonce-Generator unter Verwendung eines Zufallszahlengenerators eine Zufallszahl aufgebaut. Der Zufallszahlengenerator besteht aus zwei Prozeßsträngen, die hier als Strang 1 und Strang 2 bezeichnet werden. **Fig.** 5 zeigt die

Funktionsweise des Stranges 1, der der Hauptstrang des Zufallszahlengenerators ist.

[0071] (Kasten **301**) Strang 1 erstellt zuerst eine Datenstruktur value_list zum Abspeichern einer Liste von Zählerwerten. Die Liste ist anfänglich leer.

[0072] (Kasten **302**) Strang 1 setzt einen gegenwärtigen Zählerwert auf Null und setzt eine Flagge done_test auf FALSCH.

[0073] (Kasten **303**) Strang 1 zweigt dann Strang 2 ab. Strang 2 überträgt einen asynchronen Plattenzugriff und schläft dann, bis der Plattenzugriff abgeschlossen ist. Wenn der Plattenzugriff abgeschlossen ist, setzt der Strang 2 die Flagge done_test auf WAHR. Man beachte, daß sich Strang 1 und Strang 2 die Flagge done_test teilen.

[0074] (Kasten **304**) Strang 1 erhöht den Zählerwert um Eins.

[0075] (Kasten **305**) Strang 1 prüft dann, ob die Flagge done_test nunmehr WAHR ist, was anzeigt, daß der durch Strang 2 eingeleitete Plattenzugriff abgeschlossen ist. Wenn die Flagge done_test FALSCH ist, kehrt der Strang zum Kasten **54** zurück. Es ist daher ersichtlich, daß, während er auf den Abschluß des Plattenzugriffs wartet, der Strang 1 fortlaufend den Zählerwert erhöht.

[0076] (Kasten **306**) Wenn die Flagge done_test WAHR ist, beendet der Strang 1 den Strang 2 und sichert den Zählerwert an der ersten freien Stelle in value_list.

[0077] (Kasten **307**) Strang 1 ruft dann eine Funktion Statstest auf, die den Grad an Zufallsmäßigkeit der in value_list gesicherten Zählerwerte (oder Teile der Zählerwerte, z. B. niederwertige Bit) schätzt. Für diese Funktion kann der Chi-Square-Test, der Kolmogorov-Smirnov-Test oder der serielle Korrelationstest benutzt werden, die in [5] beschrieben sind. Die Statstest-Funktion kann optimiert werden, um sicherzustellen, daß komplizierte Berechnungen nicht für jeden Plattenzugriff wiederholt werden. Die Statstest-Funktion gibt einen Wert ab, der anzeigt, wieviele niederwertige Bit jedes gesicherten Zählerwerts als zufallsmäßig angesehen werden sollten.

[0078] (Kasten **308**) Strang 1 vergleicht den von der Statstest-Funktion abgegebenen Wert, wenn er mit der Länge der value_list kombiniert wird, mit einem vorbestimmten Schwellwert, um festzustellen, ob nunmehr genügend Zufallsbit erzeugt worden sind. Wenn nicht genügend Zufallsbit erzeugt worden sind, kehrt der Vorgang zum Kasten **302** oben zurück, um einen weiteren Zählerwert zu erstellen und zu sichern.

[0079] (Kasten **309**) Wenn die erforderliche Anzahl von Zufallsbit erzeugt worden ist, entnimmt der Strang 1 die angegebene Anzahl niederwertiger Bit aus jedem Zählerwert in der value_list und gibt diese Bitfolge als die Ausgabe-Zufallszahl ab.

[0080] Zusammengefaßt ist ersichtlich, daß vom Zufallszahlengenerator die Unvorhersagbarkeit der Zeitgabe einer Reihe von Plattenzugriffen als Quelle von Zufallsmäßigkeit bei der Erstellung von Augenblickswerten ausgenutzt wird (siehe [4]). Durch Abzweigen von neuen Strängen bei jedem Plattenzugriff werden vom Zufallszahlengenerator auch Unvorhersagbarkeiten beim Betrieb des Steuerprogramms des Betriebssystems als zweite Quelle von Zufallsmäßigkeit ausgenutzt.

[0081] Die von der Statstest-Funktion durchgeführte Analyse erlaubt dem Zufallszahlengenerator, sich selbst durch Berechnen der abzugebenden Anzahl niederwertiger Bit für jeden gesicherten Zählerwert auf einen Prozessor und eine Platte beliebiger Geschwindigkeit zu stimmen. Beispielsweise erzeugt ein System mit einer Plattenzugriffszeit mit hoher Varianz mehr Zufallsbit pro Plattenzugriff als ein System mit einer Plattenzugriffszeit mit niedriger Varianz. Beispielsweise erzeugt das System für eine Platte Quantum 1080s (6 ms Durchschnitts-Schreibzeit) und einen 66-MHz-Prozessor 486 annähernd 45 Bit pro Sekunde. Als Alternative kann die Anzahl von Bit pro Plattenzugriff hartcodiert und ein Zeitdifferenzkorrekturverfahren benutzt werden, um einen guten Grad an Zufallsmäßigkeit sicherzustellen.

[0082] Der Nonce-Generator fragt auch das Betriebssystem ab, um sicherzustellen, daß es jeden Plattenzugriff zu einer wirklichen Platte überträgt. Der abschließende Ausgabe-Augenblickswert wird durch Kombinieren der Ausgabe-Zufallszahl vom Zufallszahlengenerator mit dem Ergebnis der Abfrage der Systemparameter wie oben beschrieben unter Verwendung einer Message-Digest-Funktion gebildet.

[0083] Der oben beschriebene Nonce-Generator funktioniert am besten, wenn er auf einem Betriebssystem ausgeführt wird, das Direktzugriff zur Platte bietet, z. B. Windows 95 oder Windows NT 4.0. In einem derartigen Betriebssystem erlauben für im Kundenraum ablaufende Programme zur Verfügung stehende besondere Betriebssystemaufrufe einem Programm, den internen Puffermechanismus des Betriebssystems zu umgehen und direkt zur Platte zu schreiben. Die meisten Programme nutzen diese besonderen Betriebssystemaufrufe nicht, da sie relativ unrationell und schwer zu verwenden sind. Bei Windows 95 und Windows NT kann ein Programm diese besonderen Aufrufe nur dann benutzen, wenn das Programm auf Daten zugreift, die ein Mehrfaches der Sektorgröße der Platte sind, indem es das Betriebssystem abfragt.

[0084] Wenn das Betriebssystem keinen direkten Zugriff zur Platte bietet, dann könnte der Abfragemechanismus **24** immer noch den Zufallszahlengenerator für Plattenzugriffszeit benutzen. In diesem Fall würde die Güte der erstellten Werte sich mehr auf Unvorhersehbarkeiten im Steuerprogramm des Betriebssystems im Gegensatz zu der der Plattenzugriffszeit innewohnenden Varianz verlassen.

[0085] Bei dem oben beschriebenen Beispiel der Erfindung wird angenommen, daß das Betriebssystem einem Programm erlaubt, mehrere Stränge innerhalb eines einzigen Adreßraums abzuzweigen. Zusätzlich wird bei dem Beispiel der Erfindung ange-

nommen, daß das Betriebssystem den Strängen erlaubt, auf Synchronisierungsvariablen, wie beispielsweise Semaphore, zuzugreifen. Diese Dienste werden von den meisten modernen Betriebssystemen bereitgestellt. Im Beispiel der Erfindung werden mehrere Stränge zum Implementieren eines Mechanismus, der jede Plattenzugriffszeit quantifiziert, benutzt. Sollte jedoch eine Implementierung der Erfindung auf einem System ablaufen, das nicht mehrere Stränge oder Synchronisierungsvariablen bietet, dann könnte der Nonce-Generator diese durch andere Mechanismen, z. B. Abfrage eines physikalischen Takts, ersetzen.

Einige mögliche Abänderungen

[0086] Um die Leistung zu verbessern, kann das System mit einem verläßlichen Auditwegvalidierungsdienst verstärkt werden. Dabei wird durch den verläßlichen Dienst ein Auditweg periodisch validiert und dann eine neue Auditaufzeichnung angehängt, die Validiereraufzeichnung, die sicher für die vorhergehenden Aufzeichnungen bürgt. Beispielhafte Informationen, die in der Validiereraufzeichnung enthalten sein können, ist eine digitale Signatur des Hash-Werts aller vorhergehenden Auditaufzeichnungen auf dem Auditweg (die digitale Signatur benutzt den privaten Schlüssel des Auditvalidierungsdienstes). Von da an müssen gemietete Anwendungen nicht digitale Signaturen von Aufzeichnungen validieren, die der Validiereraufzeichnung vorangehen. Der Auditweg-Validierungsdienst könnte durch einen Dritten implementiert werden, der über ein Netz oder eine Emailverbindung zugänglich ist.

[0087] In den Verfahrensschritten von ValidateTrail [5] und [6.3] ist es möglich, daß der Zählerwert der anfänglichen Aufzeichnung nicht Null ist. In diesem Fall beginnt der Zählerwert mit Offset und Schritte [5] und [6.3] müssen im Vergleich Offset berücksichtigen.

[0088] Man beachte, daß der Verkäufer der gemieteten Anwendung der Chipkarte vertraut, Freigabe des privaten Verschlüsselungsmaterials zu vermeiden. Zusätzlich vertraut der Verkäufer der gemieteten Anwendung darauf, daß die Chipkarte ihr privates Verschlüsselungsmaterial in keinen anderen Funktionen als SignAndIncrement und GetCounter benutzt. Der Verkäufer der gemieteten Anwendung wird u. U. wünschen, den Hersteller und/oder Urheber der Personenzuordnung der Chipkarte zu validieren.

[0089] Ein Verkäufer erstellt und verteilt eine Anwendung u. U., nachdem ein Kunde eine Chipkarte erhalten hat. In diesem Fall erstellt der Verkäufer einfach eine Softwaremiet-Schlüsseldatei für den Kunden und sendet die Schlüsseldatei zum Kunden (möglicherweise nach Empfang von Bezahlung).

Universelles privates Verschlüsselungsmaterial

[0090] In einer Variante des beispielhaften Mechanismus mieten alle Kunden die Software unter Verwendung desselben Paars öffentlicher/privater Schlüssel. Hier vertraut der Verkäufer darauf, daß die Chipkarte richtig funktioniert und niemals den Wert des privaten Verschlüsselungsmaterials außerhalb der Chipkarte freigibt. Wie im Fall der **Fig.** 5 enthält die Chipkarte sowohl privates Verschlüsselungsmaterial **501** und einen Zähler **502**. Zusätzlich enthält die Chipkarte eine einmalige Seriennummer, wobei keine zwei Chipkarten dieselbe Seriennummer aufweisen. Der Schritt [1] der auf der Chipkarte implementierten Routine SignAndIncrement unterscheidet sich von der oben beschriebenen wie folgt:

[1] Berechnen der Message-Digest-Funktion von h und der Seriennummer und des Chipkartenzählers, d. h. h' ← hash(h,serial_number,counter)

[0091] Zusätzlich zu den Routinen SignAndIncrement und GetCounter stellt die Chipkarte die Routine GetSerialNumber bereit:

String GetSerialNumber()
BEGIN
[1] Abgeben der Chipkarten-Seriennummer
END

[0092] Zusätzlich sendet der Kunde/die Kundin seine/ihre Chipkarten-Seriennummer. Die Schlüsseldatei **404** enthält folgende Informationen:

- Das möglicherweise von allen Kunden geteilte universelle öffentliche Verschlüsselungsmaterial
- Die einmalige Seriennummer der Chipkarte des Kundens.

[0093] Die in der Protokolldatei abgespeicherten Hash-Aufzeichnungen berücksichtigen die Seriennummer. Beispielsweise weist die erste Hashaufzeichnung der **Fig.** 6 die folgenden Informationen für eine Nachrichtensignatur auf:

Signatur von hash(hash(96,WP...),serial_number,0)

[0094] Schritt [6] der Routine Write weist folgende Abwandlung auf:

[6] h2 ← hash(h1,serial_number,c)

[0095] Die Schritte [4] und [6.2] der Routine ValidateTrail müssen die Seriennummer benutzen (sonst würden sie stets fehlschlagen). Das von den Routinen Write und ValidateTrail benutzte Mittel zum Erhalten der Seriennummer der lokalen Chipkarte wird nicht angegeben. Man könnte beispielsweise die Chipkarte ein einziges Mal abfragen und dann die Seriennummer in einer Datei im lokalen Dateisystem abspeichern.

[0096] Ein Softwareverkäufer könnte möglicherweise eine gemietete Anwendung mit einer komplizierten Schwellwertberechnung bereitstellen. Beispielsweise könnte der Verkäufer Blöcke von 1000 Einheiten vermieten. Für jede Ablaufstunde der Software zwischen 20h00 und 6h00 am nächsten Morgen definiert die Protokollaufzeichnung eine Einheit; und für jede Stunde in einem anderen Tagesteil definiert die Protokollaufzeichnung zwei Einheiten. So könnte beispielsweise ein Kunde die Software möglicherweise 1000 Nachtstunden, 500 Tagesstunden oder ir-

gendeine berechnete Kombination von Nacht- und Tagesstunden lang benutzen.

[0097] Als Alternative kann ein Schwellwert unter Verwendung einer Booleschen Kombination mehrerer Parameter berechnet werden. Ein Softwareverkäufer könnte möglicherweise eine Mietanwendung ohne Schwellwert bereitstellen. Jeden Monat erlangt der Softwareverkäufer den aktuellen Wert der SUSL und erhält den Auditweg. Dann validiert der Verkäufer den Auditweg und berechnet eine entsprechende Gebühr. Wenn der Kunde die Software nicht weiter mieten möchte, dann bittet der Verkäufer um Rücksendung der Chipkarte. Eine Abfrage des Wertes der SUSL ist sehr leicht, wenn der Kunde die Chipkarte kurz zum Verkäufer zurücksendet. Ansonsten greift der Verkäufer über eine Netzschnittstelle auf die Chipkarte zu. Als erstes erstellt der Verkäufer eine zufallsmäßige Augenblickszahl und bittet den Kunden, die Augenblickszahl unter Verwendung von Write(nonce) dem Auditweg anzuhängen. Als nächstes wird die Routine ValidateTrail vom Verkäufer fern ausgeführt. Dieser Dienst erfordert, daß der Kunde (oder für den Kunden arbeitende Programme) die Anforderungen des Verkäufers vom Netz zur Chipkarte weitergibt. Der Verkäufer überprüft dann den validierten Auditweg für die Augenblickszahl.

Software-Kopierschutz

[0098] Der Softwaremietmechanismus kann mit einem zusätzlichen Softwarekopierschutzmechanismus verstärkt werden. Mit dem Softwarekopierschutzmechanismus wird sichergestellt, daß nur der berechtigte Kunde das Programm mieten kann. Der Softwarekopierschutzmechanismus kann sich möglicherweise eine Schlüsseldatei mit dem Softwaremietmechanismus teilen.

Geschützter Inhalt

[0099] Alle in dieser Schrift beschriebenen Mietmechanismen können auch zum Mieten von Dokumenten anstelle von Software benutzt werden. Der Ansatz besteht im Mieten eines Anzeigeprogramms, das eine Datei benutzt.

[0100] In dieser Schrift sind folgende Veröffentlichungen aufgeführt:

[1] Hardlock API, Manual Implementation of Harlock Software Protection Systems, High-Level API Version 3 Application Programming Interface, (Handimplementierung von Harlock-Softwareschutzsystemen, hohe API Version 3, Anwendungsprogrammierschnittstelle), FAST Software Security-Group, FAST-Dokument: Hohe API, Revision: 4.00e, Status 1. März 1996

[2] A. Menezes et al., Handbook of Applied Cryptography (Handbuch angewendeter Kryptographie), CRC Press, Inc. ISBN 0-8493-8523-7, S. 22–23, 224–233, 250–259, 308–311, 405–424, 433–438, 572–577, 1997

[3] R. Rivest, The MD5 message-digest algorithm (Der Message-Digest-Algorithmus MD5), RFC 1321, April 1992.

[4] P. Fenstermacher et al., Cryptographic randomness from air turbulence in disk drives (Aus Luftturbulenz in Plattenlaufwerken entstehende kryptographische Zufallsmäßigkeit), Advances in Cryptology: Crypto 1994, S. 114–120, Springer Verlag, 1994

[5] D. Knuth, The Art of Computer Programming (Die Kunst der Computerprogrammierung), Band 2, Seminumerical Algorithms Halbnumerische Algorithmen), Verlag Addison-Wesley, Reading MA, 2te Auflage, 1981, S. 38–73, ISBN 0-201-03822-6.

Patentansprüche

1. Softwaremietsystem mit mindestens einem gemieteten Programm, das einem Kunden mit einem Kundenantwortmittel mindestens einen Dienst erlaubt, wobei
– ein Abfragemittel des gemieteten Programms keinen Zugriff auf das private Verschlüsselungsmaterial eines Kundens besitzt,
– das Abfragemittel, das darin gespeichertes öffentliches Verschlüsselungsmaterial benutzt,
– unter Verwendung asymmetrischer Kryptographie, wobei das Kundenantwortmittel dem Abfragemittel des gemieteten Programms beweist, daß das Kundenantwortmittel Zugang zum privaten Verschlüsselungsmaterial des Kundens besitzt, und
– das gemietete Programm den mindestens einen Dienst für den Kunden nur dann zuläßt, wenn der Beweis erfolgreich ist.

2. Softwaremietsystem nach Anspruch 1, wobei
– das Softwaremietsystem mindestens einen Auditweg mit dem Maß, indem das gemietete Programm gemietet worden ist, sicher abspeichert,
– die Richtigkeit des mindestens einen Auditweges unter Verwendung asymmetrischer Kryptographie validiert wird.

3. Softwaremietsystem nach Anspruch 2, wobei Validierung alle folgenden sicherstellt:
– kein Auditweg ist abgeändert worden,
– kein Auditweg ist gelöscht worden, und
– die Liste aller Auditwege ist aktuell.

4. Softwaremietsystem nach Anspruch 2 oder 3, wobei das gemietete Programm nicht abläuft oder in einem begrenzten Modus abläuft, wenn die Validierung fehlschlägt.

5. Softwaremietsystem nach Anspruch 1, mit mehreren gemieteten Programmen.

6. Softwaremietsystem nach Anspruch 5, wobei das System die mehreren Programme unter Verwen-

dung sicherer aktualisierbarer Speicherstellen (Secure Updatable Storage Locations) mieten kann.

7. Softwaremietsystem nach Anspruch 6, wobei nur eine einzige sichere aktualisierbare Speicherstelle erforderlich ist, um die mehreren Programme zu mieten.

8. Softwaremietsystem nach Anspruch 2 oder 6, wobei ein erster Teil des Auditweges auf einem nicht gesicherten Medium gespeichert wird und ein zweiter Teil des Auditweges in einer sicheren aktualisierbaren Speicherstelle gespeichert wird, wobei der erste Teil des Auditweges einen Betrag gespeicherter Auditwege umfaßt.

9. Softwaremietsystem nach Anspruch 6, wobei die sichere aktualisierbare Speicherstelle von einem Kunden erlangt und zum Mieten von Software benutzt wird und danach neue gemietete Software erlangt wird, ohne eine neue sichere aktualisierbare Speicherstelle zu erfordern oder einen bestehenden Wert einer sicheren aktualisierbaren Speicherstelle rückzusetzen.

10. Softwaremietsystem nach Anspruch 5, wobei
– die gemieteten Programme dasselbe Verschlüsselungsmaterial aufweisen und
– das private Verschlüsselungsmaterial eines Kundens des Verschlüsselungsmaterials auf einer Chipkarte gespeichert ist.

11. Softwaremietsystem nach einem der vorhergehenden Ansprüche, wobei
– mindestens ein Nutzungsparameter des mindestens einen Programms auf einem Auditweg abgespeichert ist, und
– das gemietete Programm dem Kunden den mindestens einen Dienst nur dann erlaubt, wenn der mindestens eine Nutzungsparameter innerhalb eines vorbestimmten Schwellwerts liegt.

12. Softwaremietsystem nach Anspruch 11, wobei mindestens zwei Nutzungsparameter des mindestens einen Programms auf dem Auditweg abgespeichert sind.

13. Softwaremietsystem nach Anspruch 11 oder 12, wobei der mindestens eine Nutzungsparameter eine Mietzeitdauer zum Mieten des mindestens einen Programms ist.

14. Softwaremietsystem nach einem der Ansprüche 12 bis 13, wobei der mindestens eine Nutzungsparameter ein Betrag von Nutzungen des mindestens einen Programms ist.

15. Softwaremietsystem nach einem der Ansprüche 11 bis 14, wobei
– der mindestens eine Nutzungsparameter mehrere Male überprüft wird, und
– das gemietete Programm dem Kunden den mindestens einen Dienst nur dann erlaubt, wenn der mindestens eine Nutzungsparameter jedes Mal, wenn er überprüft wird, innerhalb eines vorbestimmten Zeitabstandes liegt.

16. Softwaremietsystem nach einem der obigen Ansprüche, mit einem Mietserver, der den Zugriff auf eine Chipkarte, auf der das private Verschlüsselungsmaterial eines Kundens abgespeichert ist, und/oder auf einem Auditweg, auf dem Nutzungsparameter des mindestens einen Programms abgespeichert sind, synchronisiert.

17. Softwaremietsystem nach einem der Ansprüche 11 bis 16, wobei das Programm mindestens einen Nutzungsparameter auf einem Auditweg abspeichert.

18. Softwaremietsystem nach Anspruch 17, wobei der Auditweg sicher gespeichert wird.

19. Softwaremietsystem nach einem der Ansprüche 10 bis 18, wobei die Chipkarte eine atomare Routine durchführt, die sowohl eine digitale Signatur als auch eine Operation, die über eine zeitlich veränderliche Speicherstele berechnet wird, ausführt.

20. Softwaremietsystem nach einem der obigen Ansprüche, wobei das Programm ein Anzeigeprogramm ist, das eine Datei benutzt.

21. Softwaremietsystem nach einem der obigen Ansprüche, wobei das System eine Schlüsseldatei zum Aufbewahren von öffentlichem Verschlüsselungsmaterial umfaßt.

22. Softwaremietsystem nach Anspruch 21, wobei das in der Schlüsseldatei aufbewahrte öffentliche Verschlüsselungsmaterial kryptographisch gesichert wird, wodurch es rechnerisch undurchführbar ist, irgendeinen Teil der Schlüsseldatei einschließlich des öffentlichen Verschlüsselungsmaterials ohne Änderung der Abfragemittel zu ändern.

23. Softwaremietsystem nach Anspruch 22, wobei die Schlüsseldatei Informationen enthält, die den Kunden identifizieren, dem das Programm geliefert worden ist.

24. Softwaremietsystem nach Anspruch 23, wobei die Schlüsseldatei Täuschbit zum Tarnen des darin aufbewahrten öffentlichen Verschlüsselungsmaterials umfaßt.

25. Softwaremietsystem nach einem der obigen Ansprüche, wobei das Programm kopiergeschützt ist.

26. Benutzung eines Softwaremietsystems nach Anspruch 1 bei der Verteilung eines Programms an mehrere Kunden, wobei jeder Kunde ein Softwaremietsystem nach Anspruch 1 oder 2 aufweist und wobei jeder Kunde eine identische Kopie des Programms empfängt.

27. Verfahren zum Mieten von Software mit mindestens einem gemieteten Programm, das einen Kunden mit einem Kundenantwortmittel mindestens einen Dienst erlaubt, wobei
– ein Abfragemittel des gemieteten Programms keinen Zugang zu dem privaten Verschlüsselungsmaterial eines Kundens besitzt,
– wobei das Abfragemittel darin gespeichertes öffentliches Verschlüsselungsmaterial unter Verwendung asymmetrischer Kryptographie benutzt, wobei das Kundenantwortmittel dem Abfragemittel des gemieteten Programms beweist, daß das Kundenantwortmittel Zugang zum privaten Verschlüsselungsmaterial des Kundens besitzt, und
– das gemietete Programm dem Kunden den mindestens einen Dienst nur dann erlaubt, wenn der Beweis erfolgreich ist.

28. Verfahren nach Anspruch 27, wobei
– mindestens ein Auditweg mit dem Ausmaß, in dem das gemietete Programm gemietet worden ist, sicher abgespeichert ist,
– Die Richtigkeit des mindestens einen Auditweges unter Verwendung asymmetrischer Kryptographie validiert wird.

29. Verfahren nach Anspruch 32, wobei Validierung alle folgenden sicherstellt:
– kein Auditweg ist abgeändert worden,
– kein Auditweg ist gelöscht worden, und
– die Liste aller Auditwege ist aktuell.

30. Verfahren nach Anspruch 28 oder 29, wobei das gemietete Programm nicht abläuft oder in einem begrenzten Modus abläuft, wenn die Validierung fehlschlägt.

31. Verfahren nach einem der Ansprüche 27 bis 30, mit mehreren gemieteten Programmen.

Es folgen 4 Blatt Zeichnungen

Anhängende Zeichnungen

FIG 1

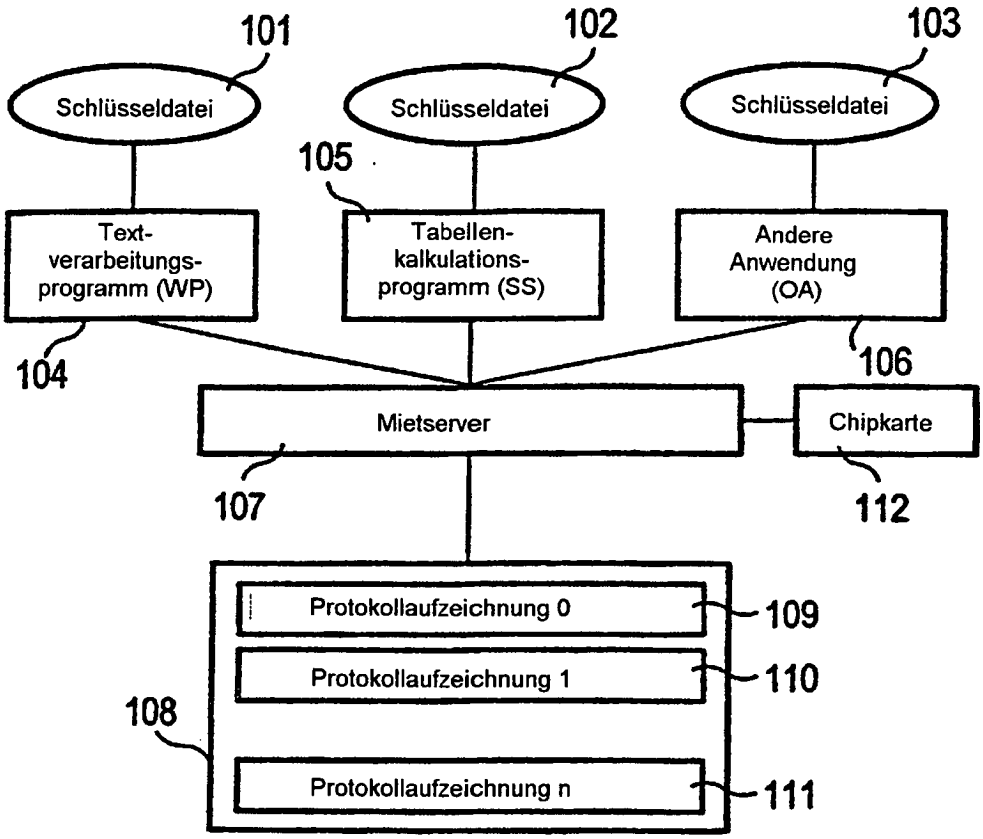


FIG 2

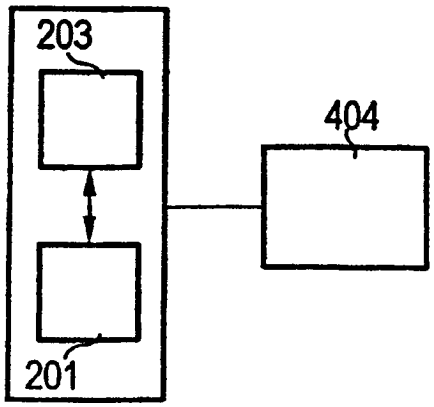


FIG 3

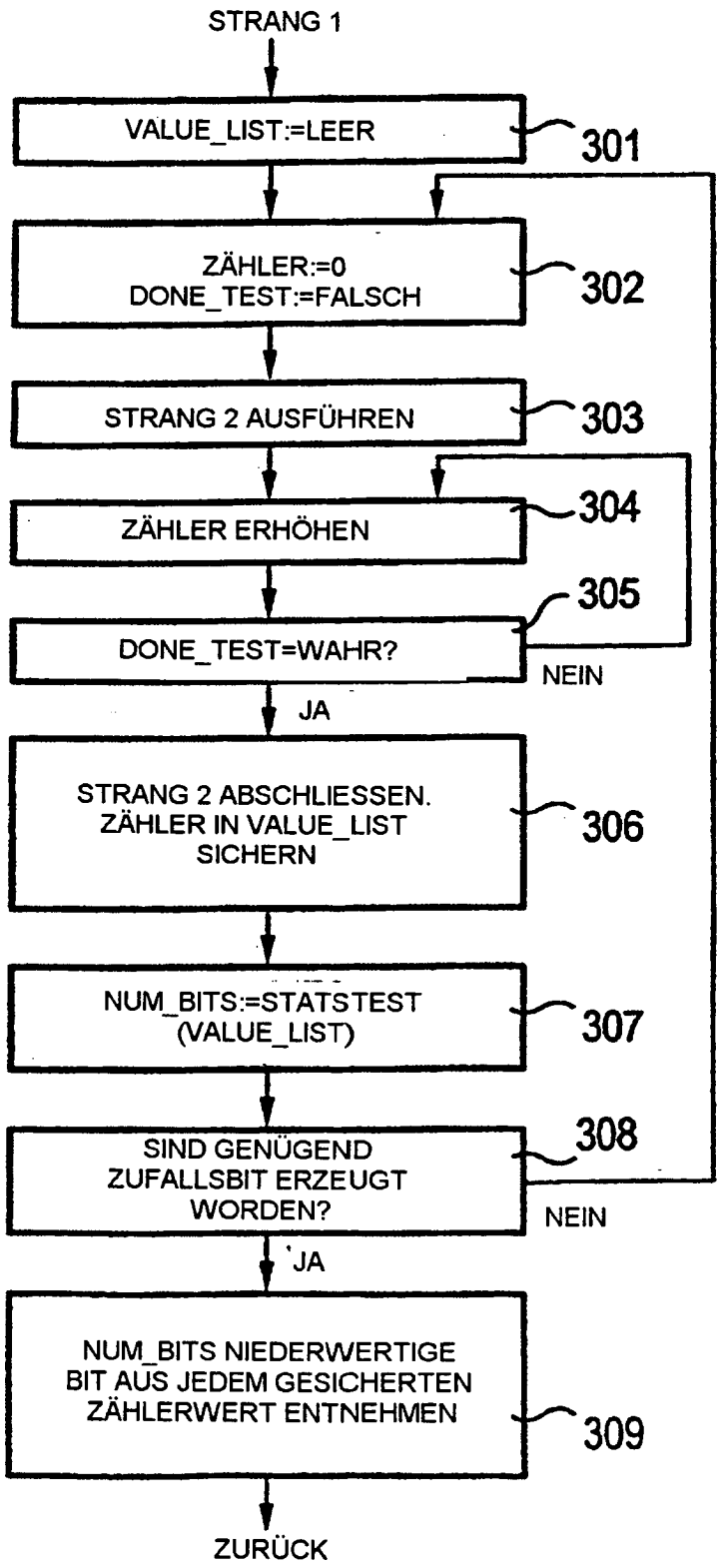

STRANG 1
VALUE_LIST:=LEER
301
ZÄHLER:=0
DONE_TEST:=FALSCH
302
STRANG 2 AUSFÜHREN
303
ZÄHLER ERHÖHEN
304
DONE_TEST=WAHR?
305
NEIN
JA
STRANG 2 ABSCHLIESSEN.
ZÄHLER IN VALUE_LIST
SICHERN
306
NUM_BITS:=STATSTEST
(VALUE_LIST)
307
SIND GENÜGEND
ZUFALLSBIT ERZEUGT
WORDEN?
308
NEIN
JA
NUM_BITS NIEDERWERTIGE
BIT AUS JEDEM GESICHERTEN
ZÄHLERWERT ENTNEHMEN
309
ZURÜCK

FIG 4

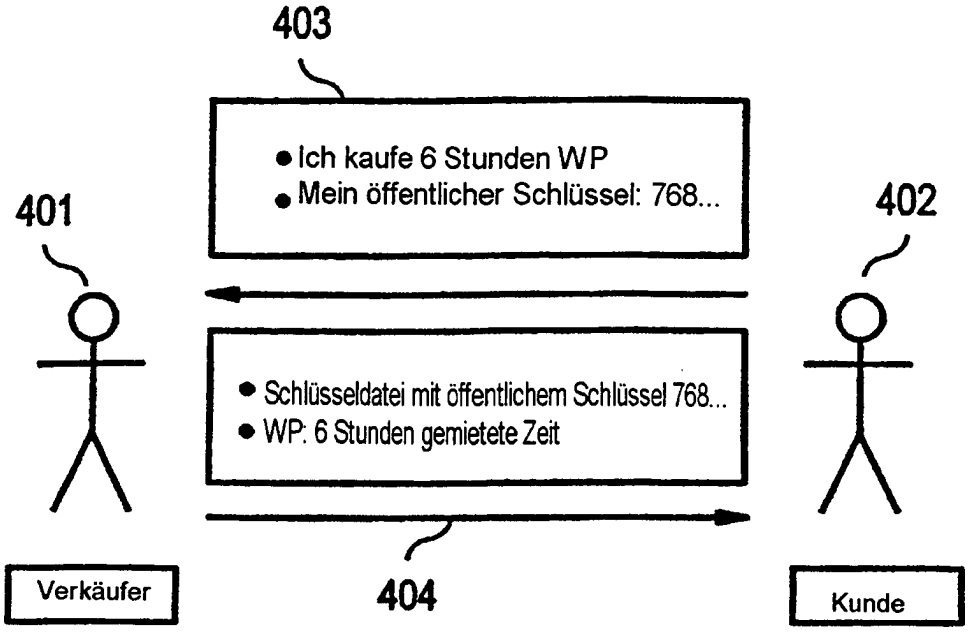

403
• Ich kaufe 6 Stunden WP
• Mein öffentlicher Schlüssel: 768...
401
402
• Schlüsseldatei mit öffentlichem Schlüssel 768...
• WP: 6 Stunden gemietete Zeit
Verkäufer
404
Kunde

FIG 5

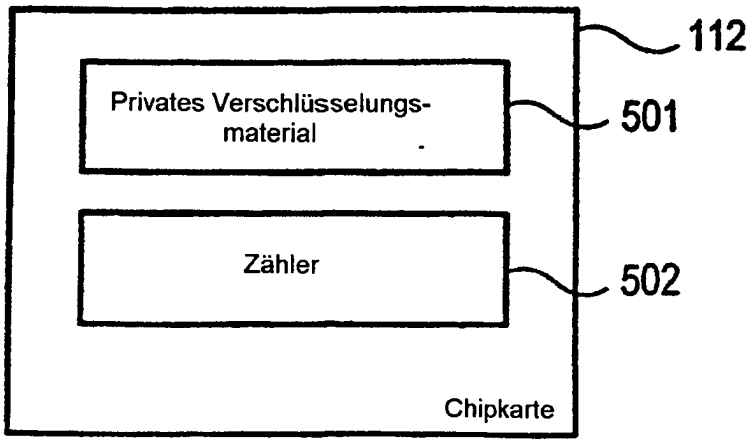
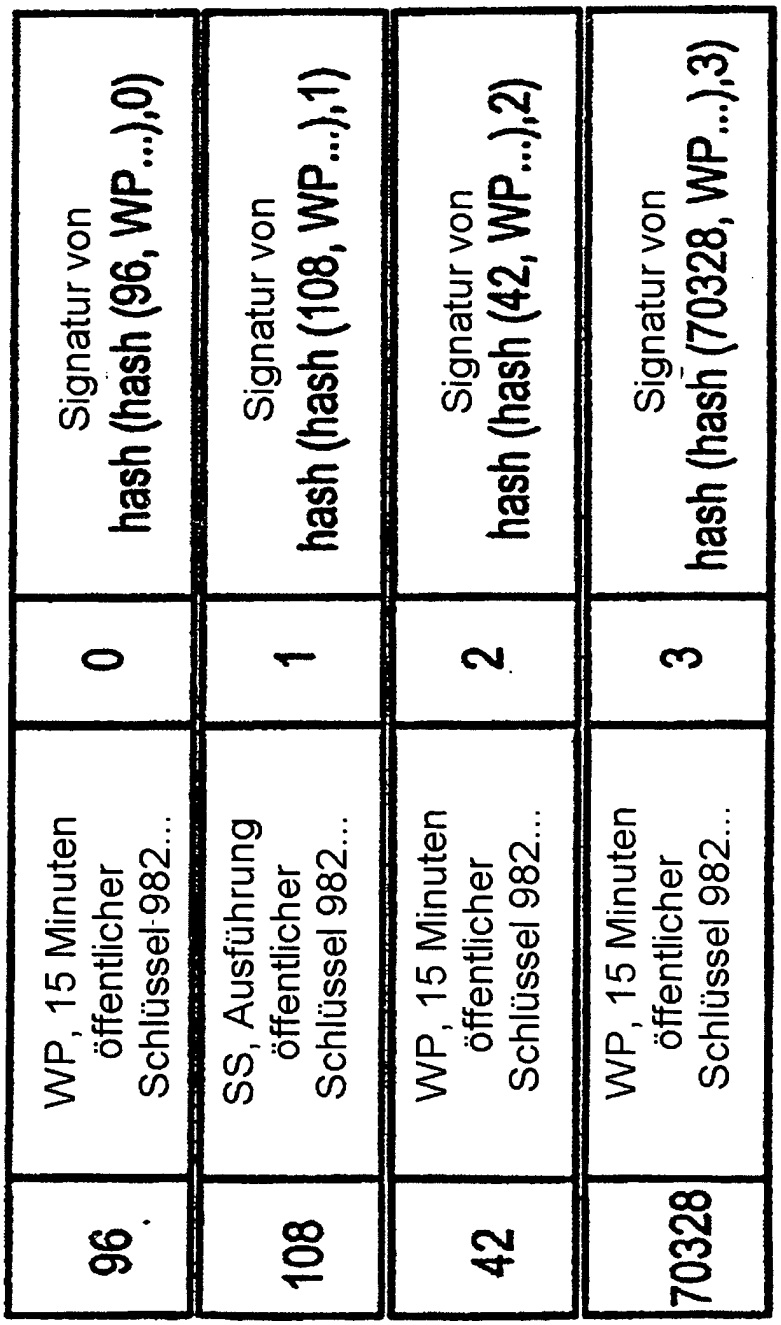

112
Privates Verschlüsselungs-material
501
Zähler
502
Chipkarte

FIG 6


96
WP, 15 Minuten öffentlicher Schlüssel 982...
0
Signatur von hash (hash (96, WP...),0)
108
SS, Ausführung öffentlicher Schlüssel 982...
1
Signatur von hash (hash (108, WP...),1)
42
WP, 15 Minuten öffentlicher Schlüssel 982...
2
Signatur von hash (hash (42, WP...),2)
70328
WP, 15 Minuten öffentlicher Schlüssel 982...
3
Signatur von hash (hash (70328, WP...),3)