(19) Deutsches Patent- und Markenamt

(10) **DE 10 2022 000 967 A1** 2023.09.21

(12) # Offenlegungsschrift

(21) Aktenzeichen: **10 2022 000 967.6**
(22) Anmeldetag: **19.03.2022**
(43) Offenlegungstag: **21.09.2023**

(51) Int Cl.: ***G06F 21/64*** (2013.01)
G06F 21/60 (2013.01)
G06F 16/00 (2019.01)
G06Q 50/18 (2012.01)

(71) Anmelder:
Datalisk GmbH, 90409 Nürnberg, DE

(72) Erfinder:
Erfinder wird später genannt

(56) Ermittelter Stand der Technik:

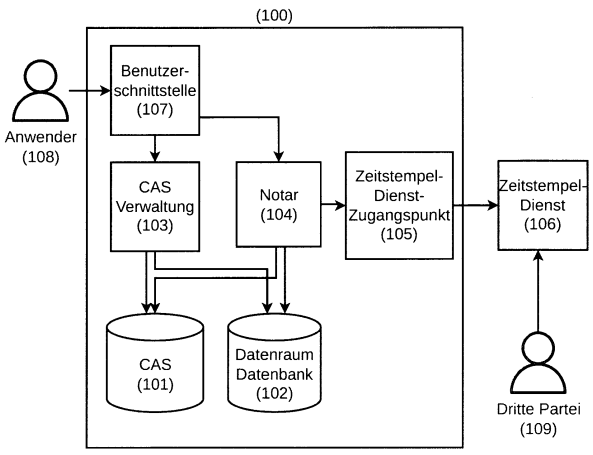
DE	**10 2021 003 888**	**A1**
US	**2021 / 0 064 580**	**A1**
EP	**3 761 206**	**A1**
WO	**2006/ 113 490**	**A2**
WO	**2021/ 099 627**	**A1**

Prüfungsantrag gemäß § 44 PatG ist gestellt.

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen.

(54) Bezeichnung: **Verfahren zur Speicherung von Daten die in einer Baumstruktur organisiert und in einem Speicher eines Computersystems abgespeichert sind, wobei der Speicher wiederkehren einen Hash des aktuellen Wurzel-Objekts der Baumstruktur über einen Zeitstempel-Dienst absichert**

(57) Zusammenfassung: Um eine kontinuierliche, nachprüfbare und revisionssichere Datenspeicherung mithilfe von Computer-Systemen bereitzustellen, wird ein Verfahren zur Speicherung von Daten, die in einer Baumstruktur organisiert und in einem Speicher eines Computersystems abgespeichert sind, insbesondere von in einer Verzeichnisstruktur organisierten Dateien, vorgeschlagen, wobei der Speicher einzelne Daten-Objekte über einen Hash ihres Inhalts referenziert, und wiederkehrend einen Hash des aktuellen Wurzel-Objekts der Baumstruktur über einen Zeitstempel-Dienst absichert.



Beschreibung

[0001] Verfahren zur Speicherung von Daten die in einer Baumstruktur organisiert und in einem Speicher eines Computersystems abgespeichert sind, wobei der Speicher wiederkehrend einen Hash des aktuellen Wurzel-Objekts der Baumstruktur über einen Zeitstempel-Dienst absichert.

[0002] Unter „Daten“ sind insbesondere Dateien und Verzeichnisse zu verstehen, wie sie aus gebräuchlichen Computer-Systemen bekannt sind. Als revisionssicher kann ein Datenspeicher betrachtet werden wenn dieser insbesondere folgende Eigenschaften aufweist:

a) Es kann nachgewiesen werden, dass eine bestimmte Datei zu einem bestimmten Zeitpunkt existierte. Insbesondere kann nachgewiesen werden, dass keine Daten rückdatiert wurden, um z.B. den Eindruck zu erwecken, dass eine Datei früher erstellt wurde als es tatsächlich der Fall war.

b) Es kann nachgewiesen werden, dass ein bestimmtes Verzeichnis (einschließlich aller darunter liegenden Inhalte) zu einem bestimmten Zeitpunkt existierte.

c) Es können Änderungen oder Löschungen von Dateien und Verzeichnissen nachvollzogen werden.

d) Es kann nachgewiesen werden, dass keine Daten aus der aufgezeichneten Historie entfernt wurden, also z.B. dass die Existenz einer bestimmten Datei in der Vergangenheit nicht nachträglich verborgen werden kann.

Stand der Technik

[0003] Es sind Verfahren bekannt, um Daten durch eine Art manipulationssicheres Siegel auf eine einzelne Datei, einem Zeitpunkt zuordnen zu können und vor Veränderung zu schützen bzw. solch eine Manipulation erkennen zu können. Hierbei sind insbesondere digitale Signaturen mit Zeitstempel (z.B. basierend auf RFC 3161 - u.a. angeboten von der Bundesdruckerei) zu nennen. Komplette Verzeichnisse können hiermit über den Umweg eines Archivs (z.B. ZIP-Archiv) revisionssicher gespeichert werden.

[0004] Ein „Merkle Tree“ (beschrieben in Patent US4309569) ist eine Datenstruktur mithilfe derer eine beliebige Anzahl von Datenelementen zu einem einzigen Hash (deutsch: „Prüfsumme“) verdichtet werden kann. Insbesondere erlaubt die Datenstruktur die Erstellung eines „Merkle Proof“ durch den kompakt belegt werden kann, dass ein bestimmtes Datenelement Bestandteil des Tree ist, ohne den kompletten Tree zur Verfügung stellen zu müssen.

[0005] Basierend auf Merkle-Trees ist 2010 die erste Blockchain-Lösung „Bitcoin“ veröffentlicht worden, welche eine revisionssichere Datenablage ermöglicht. Die aktuell verbreitetsten Blockchain-Lösungen Bitcoin und Ethereum erreichen durch die Größe ihrer globalen Verteilung ein äußerst hohes Maß an Revisionssicherheit, sind jedoch anhand extrem hoher Speicherkosten nur für äußerst kleine Datenmengen geeignet.

[0006] Das 2012 erschienene Verfahren „OpenTimestamps“, sowie Patent CN106815530B, versuchen die Kostenproblematik der direkten Datenablage in Blockchain zu lösen, indem sie Hashes der zu speichernden Dateien erstellen, diese durch Nutzung eines Merkle-Tree erneut verdichten zu einem einzelnen Hash, welcher letztlich in der Blockchain „Bitcoin“ gespeichert wird. Anwender können bei OpenTimestamps im Anschluss eine sog. „Proof-Datei“ erzeugen die beweist, dass ihre Datei Teil des Verdichtungsprozess war. Diese Proof-Datei ist dauerhaft durch den Anwender zu speichern.

[0007] Weiterhin sind diverse Verfahren unter dem Namen „Content Addressed Storage“ bekannt. In einem solchen Datenspeicher werden Daten-Objekte über ihren Inhalt (z.B. ihrem Hash) adressiert. Soll der Inhalt eines Objekts geändert werden, ergibt sich aus der Änderung zwangsläufig ein neues Objekt, da sich hierdurch dessen Adresse (ihr Hash) ändert. Durch entsprechende Referenzierung zwischen den einzelnen Daten-Objekten lassen sich somit komplexe unveränderliche Datenstrukturen bilden. Hierdurch ergeben sich Vorteile beim Auffinden von Daten und im Einsatz in verteilten System. Ein Nachteil ist, dass schreibende Operationen meist aufwändiger sind. Bekannte Verfahren sind „Content-addressed file systems“ (1972, Coulouris, G. F.;, „Towards Content-Addressing in Data Bases“, The Computer Journal), sowie das Versionskontrollsystem Git (2005, Linus Torvalds).

Problemstellung

[0008] Der Erfindung liegt das Problem zugrunde, auf eine ökonomische Weise eine kontinuierliche, beweisbar revisionssichere Versionshistorie für einzelne Dateien und komplette Verzeichnisse gewährleisten zu können. Aktuelle Verfahren arbeiten dagegen meist vertrauensbasiert, z.B. durch Vertrauen in das Versprechen eines unabhängigen IT-Betrieber, eine revisionssichere Datenspeicherung zu gewährleisten.

[0009] Anwendungsbeispiel 1: Bei der Speicherung von steuerlich relevanten Belegen (u.a. Rechnungen) ist es erforderlich, diese so zu speichern, dass

nachträgliche Modifikationen nicht möglich sind und keine Belege nachträglich entfernt werden können (siehe GoBD-Richtlinie, Bundesministerium der Finanzen, vom 28.11.2019).

[0010] Anwendungsbeispiel 2: Beschlüsse einer Gesellschafterversammlung sollen beweiskräftig archiviert werden. Wurde festgelegt, dass alle Beschlussdokumente einer Gesellschaft in einem Verzeichnis abgelegt werden, kann anhand einer Versionshistorie des Verzeichnisses eine vollständige Historie der Gesellschafterbeschlüsse belegt werden. Hierdurch ist auch eine nachträgliche Löschung einzelner Beschlüsse ausgeschlossen. Insbesondere ist dies für alleinige Gesellschafter-Geschäftsführer interessant.

[0011] Anwendungsbeispiel 3: Bei Verhandlung über den Kauf von Unternehmens anteilen, hat der Verkäufer oft umfangreich über Werte und Risiken zu informieren. Über eine revisionssichere Datenspeicherung kann der Verkäufer zu einem späteren Zeitpunkt einfach (und unabhängig von einem dritten Dienstleister) belegen, dass alle relevanten Informationen zur Verfügung gestellt worden sind. Der Käufer kann ggf. belegen, das bestimmte Informationen gefehlt haben.

[0012] Der beschriebene Bedarf nach revisionssicherer digitaler Datenhaltung besteht insbesondere seit dem in großem Umfang Dokumente digital ausgetauscht werden, also seit mindestens 20 Jahren. Eine direkte Datenspeicherung in einer Blockchain wie Bitcoin oder Ethereum ist zu kostenintensiv und wenig performant. Ferner werden Daten dort öffentlich und weltweit verteilt gespeichert, was bei einer direkten Datenspeicherung in vielen Fällen datenschutzrechtlich problematisch wäre. Die bekannten Verfahren der digitalen Signatur und ähnliche Verfahren wie OpenTimestamps erlauben lediglich einzelne Dateien revisionssicher zu speichern. Die Umsetzung eines revisionssicheren Datenspeichers (wie anfangs definiert) ist mit diesen Verfahren nur äußerst aufwendig umsetzbar. Soll ein revisionssicherer Datenspeicher mit diesen Verfahren umgesetzt werden wäre eine denkbare Umsetzung, dass regelmäßig ein ZIP-Archiv des Verzeichnisses, welches den revisionssicheren Datenspeicher enthält, erstellt wird. Dies führt jedoch zu einem sehr hohen Speicherverbrauch, da jedes Archiv erneut den kompletten Datenbestand enthält. Daher ist der Einsatz von digitalen Signaturen o.ä. für die Umsetzung eines revisionssicheren Datenspeichers in den meisten Fällen nicht praktikabel. Soll es im Sinne des Patentanspruch 2 ermöglicht werden einzelne Unterverzeichnisse und Dateien als echt zu belegen, ohne unnötig Daten preisgeben zu müssen, wäre das Verfahren analog auf Unterordner und Dateien anzuwenden. Hierdurch ergibt sich sogar ein exponentiell wachsender Speicherverbrauch.

Darstellung der Erfindung

[0013] Der Erfindung liegt die Aufgabe zugrunde Daten, insbesondere Dateien und Verzeichnisse, kontinuierlich, nachprüfbar und revisionssicher zu Speichern. Digitale Signaturen o.ä. sind geeignet um einzelne Dateien/Dokumente einem Zeitpunkt zuordnen zu können. Diese Verfahren eigenen sich jedoch nicht für einen revisionssicheren Datenspeicher, insbesondere aufgrund des stark und ggf. sogar exponentiell steigenden Speicherverbrauchs.

[0014] Dieses Problem wird durch ein Verfahren gelöst welches, durchführbar mit einem Computersystem mit einem Speicher, zur Speicherung von Daten die in einer Baumstruktur organisiert und in einem Speicher eines Computersystems abgespeichert sind, insbesondere von in einer Verzeichnisstruktur organisierten Dateien, wobei der Speicher einzelne Daten-Objekte über einen Hash ihres Inhalts referenziert und wiederkehrend einen Hash des aktuellen Wurzel-Objekts der Baumstruktur über einen Zeitstempel-Dienst absichert.

[0015] Die Bezeichnung „Hash“ (plural: „Hashes“) bezeichnet eine kryptographische Prüfsumme eines Inhalts. Bevorzugt wird für die Generierung eines Hash eine allgemein als sicher anerkannte kryptografische Einweg-Hashfunktion genutzt, welche insbesondere eine starke Kollisionsresistenz aufweist. Die Wahrscheinlichkeit, dass zwei ungleiche Inhalte denselben Hash besitzen ist vernachlässigbar, wenn eine ausreichende Hash-Länge verwendet wird. Daten-Objekte referenzieren andere Daten-Objekte von denen sie logisch abhängig sind. Daten-Objekte werden über einen Hash ihres Inhalts im Speicher referenziert. Hierdurch ergibt sich eine Baumstruktur von Daten-Objekten. Das Daten-Objekt an der obersten Stelle der Baumstruktur, welches also von keinem anderen Daten-Objekt referenziert wird, wird als Wurzel-Objekt bezeichnet. Die initiale Befüllung der Baumstruktur mit Daten-Objekten sowie eine Änderung der Baumstruktur werden als „Commit“ bezeichnet. Ein Commit besteht aus einer Menge von neuen Daten-Objekten wovon eines ein Wurzel-Objekt ist. Bevorzugt werden Commits sequenziell durchnummeriert. Der beschriebene Speicher wird im folgenden als CAS (Content-Addressed-Storage) bezeichnet.

[0016] Bevorzugt wird das CAS zur Speicherung von Dateien die in einer Verzeichnisstruktur organisiert sind genutzt. Dabei sind die bevorzugten Ausprägungen der Daten-Objekte des CAS einerseits Datei-Objekte, welche eine einzelne Datei z.B. anhand ihres Namens und binären Dateiinhalts beschreiben. Ferner Verzeichnis-Objekte welche ein einzelnes Verzeichnis beschreiben, z.B. anhand des Verzeichnisnamens und enthaltene Verzeichnis- und Datei-Objekte referenziert. Handelt es sich bei

einem Verzeichnis-Objekt um das Wurzel-Objekt, wird es auch als Wurzel-Verzeichnis bezeichnet. Alternativ kann das CAS u.a. zur Speicherung von Emails genutzt werden.

[0017] Da der Hash eines Daten-Objekts von dessen Inhalt abhängig ist, bewirkt ein anderer Inhalt zwangsläufig ein separates Datenobjekt. Diese unveränderliche Natur der Daten-Objekte des CAS bedeutet, dass bestehende Daten-Objekte nicht modifiziert/manipuliert werden können, ohne dass dies auch eine Änderung des Hash des Wurzel-Objekts bewirken würde, wodurch eine Datenmanipulation einfach und sicher erkannt werden kann.

[0018] Der Hash des Wurzel-Objekts soll über einen Zeitstempel-Dienst abgesichert werden. Über einen Zeitstempel-Dienst kann ein beliebiger „Wert“ abgesichert werden (nachfolgend wird dieser Vorgang als „absichern“ oder „Absicherung“ bezeichnet). Dabei kann ein Wert beliebige Daten darstellen, z.B. den Hash des Wurzel-Objekts. Durch eine Absicherung kann bewiesen werden, dass ein Wert zu einem gegebenen Zeitpunkt bekannt war. Das Ergebnis der Absicherung ist auf eine Weise zu speichern, dass dieses nachträglich wieder aufgefunden und der Bezug zu dem abgesicherten Wert erhalten bleibt, bevorzugt über eine Datenbank. Somit kann für einen abgesicherten Wert bewiesen werden ab welchem Zeitpunkt dieser bekannt war. Durch eine Absicherung des Hash des Wurzel-Objekts kann gezeigt werden, dass alle Daten-Objekte des CAS von denen das Wurzel-Objekt abhängt zu diesem Zeitpunkt existiert haben müssen. Bei dem Wert handelt es sich bevorzugt unmittelbar um den Hash des Wurzel-Objekts. Alternativ oder zusätzlich kann es vorteilhaft sein u.a. zusätzliche Informationen mit dem Hash des Wurzel-Objekts abzusichern oder beispielsweise den Hash des Wurzel-Objekts indirekt über einen Merkle-Tree abzusichern.

[0019] Mögliche Zeitstempel-Dienste wären u.a. zentrale Signatur-Dienste basierend auf RFC 3161 und Blockchain-Computersysteme. Ein bevorzugter Zeitstempel-Dienst ist die öffentliche Blockchain „Ethereum“ welche durch ein global verteiltes Computersystem implementiert ist. Bei Nutzung von einer Blockchain ist das Ergebnis der Absicherung bevorzugt ein Datensatz, der den abgesicherten Wert mit der Stelle in der Blockchain, an dem der Wert hinterlegt wurde, in Verbindung bringt. Im Fall von Ethereum entspricht die „Stelle in der Blockchain“ bevorzugt einer „Ethereum Transaction ID“ welche die „Ethereum Transaction“ angibt in welcher der Wert gespeichert wurde.

[0020] Der Hash des jeweils aktuellen Wurzel-Objekts wird wiederkehrend über den Zeitstempel-Dienst abgesichert. Hierdurch lassen sich Änderungen im CAS einem Zeitfenster zuordnen. Dies gelingt, indem für ein gegebenes Daten-Objekt gezeigt wird, dass er erstmals zu einem bestimmten Zeitpunkt erschienen ist (über ein abgesichertes Wurzel-Objekt) und zum Zeitpunkt einer vorherigen Absicherung noch nicht vorhanden war. Falls Daten-Objekte durch ein zeitlich späteres Wurzel-Objekt nicht mehr referenziert werden, kann dementsprechend gezeigt werden, dass Inhalte aus dem CAS entfernt wurden. Dieses Zeitfenster kann durch häufigeres oder selteneres Absichern je nach Anwendungsfall verkürzt oder vergrößerst werden.

[0021] Ein wesentlicher Vorteil der mit der Erfindung erzielt wird besteht darin, dass ohne einen erheblichen Speicherverbrauch eine revisionssichere Speicherung eines beliebig großen und komplexen Datenbestands ermöglicht wird. Die Nutzung des CAS ermöglicht hierbei, bei Änderungen Daten nicht unnötig duplizieren zu müssen, wie in dem vorhergehenden Beispiel basierend auf dem Stand der Technik dargelegt. Da in dem CAS für alle Daten-Objekte stets Hashes zu Verfügung stehen, kann stattdessen effizient auf bestehende Daten anhand ihrer Hashes referenziert werden. Das CAS ermöglicht eine hoch effiziente Verdichtung der Daten zu einem einzelnen Hash. Lediglich dieser Hash ist wiederkehrend über einen Zeitstempel-Dienst abzusichern, was jedoch genügt um die Revisionssicherheit eines großen Datenbestands zu garantieren und erhebliche Kostenvorteile gegenüber einer direkte Speicherung von Daten-Objekten beispielsweise in einer Blockchain ergibt.

[0022] Ebenfalls ist eine nachträgliche Prüfung der Revisionssicherheit effizient möglich, da Daten nicht in mehrfacher Ausführung gespeichert wurden und wieder verglichen werden müssen. Sofern ein Zeitstempel-Dienst mit hohem Sicherheitsniveau genutzt wird, ist davon auszugehen, dass eine Datenmanipulation mit Sicherheit erkannt werden kann. Die Beweiskraft ist damit meist Schriftstücken in Papierform überlegen, da bei diesen eine Rück- oder Vordatierung durch erneutes ausdrucken oft problemlos möglich ist.

[0023] Aus der Nutzung des bevorzugten Zeitstempel-Dienstes Ethereum o.ä. Blockchain-Computersysteme ergibt sich der Vorteil, dass die Revisionssicherheit des Datenspeichers nicht vertrauensbasiert ist, sondern vollständig auf einem technischen Verfahren basiert. Somit kann eine vollständig nachprüfbare Revisionssicherheit erzielt werden, die insbesondere gegenüber Dritten belegbar ist. Für den Eigentümer der Daten entsteht keine dauerhafte Abhängigkeit z.B. zu einem zentralen IT-Dienstleister, wie dies bei den vertrauensbasierten Verfahren der Fall wäre, wodurch sich bei einer langfristigen Archivierung von Daten meist ein erheblicher Kostenvorteil ergibt. Ferner kommen dezentrale Systeme wie Ethereum ohne Vertrauen in Zertifikate

aus und bieten einen Schutz vor einer möglichen Kompromittierung eines Signatur-Dienstes z.B. in Folge eines Hacker-Angriffs.

[0024] Mit der zuvor offenbarten Erfindung kann eine kontinuierliche, nachprüfbare und revisionssichere Datenspeicherung erreicht werden. Die eingangs dargestellten Anwendungsbeispiele sind mit einem solchen Datenspeicher also effizient und auf sehr hohem Sicherheitsniveau umsetzbar.

[0025] Bevorzugt ist vorgesehen, dass der Hash über das SHA3-256 Verfahren erzeugt wird.

[0026] Das Verfahren SHA3-256 gilt allgemein als modern und äußerst sicher und ist daher älteren Hash-Verfahren vorzuziehen.

[0027] Bevorzugt ist vorgesehen, dass die Absicherung über den Zeitstempel-Dienst in regelmäßigen zeitlichen Abständen erfolgt.

[0028] Würde ein Angreifer nachträglich die Existenz eines Daten-Objekts verbergen wollen, müssten mindestens alle nachfolgenden Wurzel-Objekte und Absicherungen aus dem Speicher entfernt werden, die dieses Daten-Objekt referenzieren. Diese Art der Datenmanipulation kann erkannt werden, wenn festgelegt wurde, dass eine Absicherung in regelmäßigen zeitlichen Abständen erfolgen muss. Beispiele hierfür sind „stündlich“ oder „jeden Tag um 00:00 Uhr bis auf Sonn- und Feiertage“. Bevorzugt ist eine tägliche Absicherung, um 00:00 Uhr. Würde eine Absicherung zu einem erwarteten Zeitpunkt, unter Berücksichtigung einer gewissen Toleranz, fehlen kann somit auf eine mögliche Manipulation geschlossen werden. Keine zeitliche Regelmäßigkeit liegt vor, wenn beispielsweise eine Absicherung „nach einer Änderung innerhalb von 24 Stunden zu erfolgen hat“, da hier die Regel von anderen Faktoren als von der Zeit abhängt und daher für Dritte nicht einfach feststellbar ist, ob eine Absicherung ausgelassen wurde.

[0029] Bevorzugt ist vorgesehen, dass bei einer Datenänderung im Speicher ein neues Wurzel-Objekt ein vorheriges Wurzel-Objekt referenziert.

[0030] Hierdurch ist die zeitliche Abfolge von Änderungen direkt im CAS festgeschrieben. Eine Historie von Änderungen lässt sich somit direkt aus den Daten des CAS ableiten, was meist geringeren Rechenaufwand bedarf. Außerdem ist es nicht mehr möglich einen Commit, der selbst nicht abgesichert wurde, nachträglich zu entfernen, wenn dieser bereits über ein nachfolgendes abgesichertes Wurzel-Objekt referenziert wird. Insofern bedeutet dies eine Kostenreduktion, da man insgesamt mit weniger Absicherungen auskommen kann.

[0031] Bevorzugt ist vorgesehen, dass für ein im Speicher neu hinzugefügtes Daten-Objekt dessen Hash in einem Merkle-Tree gespeichert wird, und der Hash der Spitze des Merkle-Tree über den Zeitstempel-Dienst abgesichert wird.

[0032] Als Merkle-Tree eignen sich alle gängigen Implementierungen, bevorzugt basierend auf Patent US4309569A. Unter Merkle-Proof ist eine Funktion zu verstehen, welche für eine gegebenes Element eine erwartete Spitze des Merkle-Tree errechnet, womit sich die Zugehörigkeit des Elements zu einem Merkle-Tree belegen lässt. Eine Baumregion besteht aus einem gegebenen Daten-Objekt und aller Daten-Objekte von denen dieses direkt oder indirekt abhängt. Somit ermöglicht die Absicherung des Hashes eines Daten-Objektes für dessen Baumregion zeigen zu können, dass diese zum Zeitpunkt der Absicherung existiert hat. Bevorzugt werden die Hashes aller Daten-Objekte eines Commits abgesichert. Die Absicherung eines Hashes mithilfe des Zeitstempel-Dienstes ist jedoch meist (sowohl bei Ethereum, als auch bei Signatur-Dienstleistern) mit hohen Kosten verbunden, insbesondere bei intensiver Nutzung des Datenspeichers. Um hohe Kosten zu vermeiden, werden in einem weiteren Schritt alle abzusichernden Hashes über einen Merkle-Tree zu einem einzelnen Hash verdichtet. Wie zuvor offenbart, wird der Hash des jeweils aktuellen Wurzel-Objekts wiederkehrend über einen Zeitstempel-Dienst abgesichert. Wenn die Absicherung des Wurzel-Objekts durchgeführt werden soll, werden nun dieser und alle weiteren abzusicherenden Hashes in einen Merkle-Tree eingefügt und die Spitze des Merkle-Tree wird über einen Zeitstempel-Dienst abgesichert, wodurch auch alle Hashes die Teil des Merkle-Tree sind indirekt abgesichert werden. Bevorzugt ist der Wert der abgesichert wird unmittelbar die Spitze des Merkle-Tree.

[0033] Dass ein gegebener Hash „h“ zu einem bestimmten Zeitpunkt „t“ existiert hat ist bewiesen, wenn ein Merkle-Proof bekannt ist der zeigt, dass „h“ eine Spitze des Merkle-Tree „m“ ergibt, und dass ein Ergebnis einer Absicherung bekannt ist, welches zeigt, dass „m“ zum Zeitpunkt „t“ existiert hat.
Bevorzugt wird der erzeugte Merkle-Tree gespeichert und ein benötigter Merkle-Proof daraus bei Bedarf erzeugt. Hierfür kann der erzeugte Merkle-Tree zusammen mit dem Ergebnis der Absicherung, dem Hash des Wurzel-Objekt und dem aktuellen Datum in einer Datenbank gespeichert werden. Soll für „h“ ein Beweis geführt werden, wird für „h“ das dazugehörige Wurzel-Objekt gesucht. Wenn für dieses Wurzel-Objekt oder einem der nachfolgenden Wurzel-Objekte ein Merkle-Tree gefunden wird, der „h“ enthält, kann hierfür der benötigte Merkle-Proof erstellt werden. Falls kein Merkle-Tree gefunden wird der „h“ enthält, kann kein Beweis erzeugt wer-

den und es weißt auf eine mögliche Manipulation hin. Alternativ zum dargelegten bevorzugten Vorgehen ist es u.a. denkbar alle möglichen Merkle-Proofs im Voraus zu erstellen und diese zu Speichern. Hieraus ergibt sich jedoch ein höherer Speicherverbrauch.

[0034] Somit kann auch für eine einzelne Baumregion gegenüber Dritten bewiesen werden ob diese zu einem bestimmten Zeitpunkt bekannt war, ohne hierfür Daten-Objekte außerhalb der Baumregion ebenfalls zur Verfügung stellen zu müssen. Beispielweise kann bei der Nutzung des CAS zu Abbildung einer Verzeichnisstruktur, ein Unterverzeichnis exportiert werden und dessen Revisionssicherheit belegt werden, ohne Daten anderen Verzeichnissen preisgeben zu müssen. Ferner lässt der Merkle-Proof keine Rückschlüsse auf den Inhalt anderer Daten-Objekte zu. Hierdurch ermöglicht die Ausgestaltung einen hohen Grad an Datensparsamkeit.

[0035] Bevorzugt ist vorgesehen, dass Daten-Objekte vorhergehende Daten-Objekte über deren Hash referenzieren.

[0036] Indem ein Daten-Objekt „x“ ein vorhergehendes Daten-Objekt referenziert, wird eine Historie für „x“ im CAS festgeschrieben. Das Daten-Objekt referenziert folglich die für die Bestimmung seiner eigenen Historie erforderlichen Daten.
Dies ermöglicht es dritten Parteien Zugriff auf eine Baumregion zu geben, und befähigt diese eine Historie zu ermitteln, ohne hierfür Daten-Objekte außerhalb der Baumregion ebenfalls zur Verfügung stellen zu müssen. Insofern der Hash des Daten-Objekt „x“ und die Hashes seiner vorhergehenden Daten-Objekte abgesichert wurden, bevorzugt über das zuvor offenbarte Verfahren mithilfe eines Merkle-Tree, kann ebenfalls die Revisionssicherheit der Historie belegt werden, ohne hierfür Zugriff auf Daten-Objekte außerhalb der Baumregion haben zu müssen. Ferner bedeutet die Tatsache, dass ein Daten-Objekt die für die Bestimmung seiner eigenen Historie erforderlichen Daten selbst referenziert, dass die Historie in geringerem Ausmaß ausgehend von einem Wurzel-Objekt interpretiert werden muss. Somit ergibt sich eine Historie durch klar nachvollziehbare Referenzen, anstatt eines komplexen Algorithmus, was die Revisionssicherheit einfacher nachvollziehbar und robuster macht.
Bevorzugt sollten daher alle Daten-Objekte vorhergehende Daten-Objekte referenzieren soweit sich für ein bestimmtes Daten-Objekt ein sinnvoller Vorgänger bestimmen lässt.

[0037] Bevorzugt ist vorgesehen, dass vorbestimmte Inhalte, bevorzugt Name, Datum, Dateiinhalt, der Daten-Objekte als eigenständige Daten-Objekte über einen Hash referenziert werden.

[0038] Die zuvor beschriebene unveränderliche Natur von Daten-Objekten in einem CAS bewirkt, dass eine nachträgliche Modifikation des Inhalts eines Daten-Objekts nachträglich entdeckt werden kann. Bevorzugt werden sensitive Inhalte wie Name, Datum, Dateiinhalt in separate Daten-Objekte, genannt Inhalts-Objekte, ausgelagert.

[0039] Die Speicherung eines Inhalts in einem separaten Daten-Objekt ermöglicht es, dass der gespeicherte Inhalt des Daten-Objekts geändert oder gelöscht werden kann, wodurch dieses Daten-Objekt zwar ungültig wird (der Hash stimmt nicht mehr mit dem Inhalt überein), andere Daten-Objekte die auf dieses verweisen werden jedoch nicht unmittelbar beeinträchtigt. Dies ermöglicht es punktuell Daten-Objekte permanent zu löschen, ohne die Revisionssicherheit anderer Daten-Objekte zu beeinträchtigen.

[0040] Somit kann beispielsweise der Name einer Datei dauerhaft aus der gesamten Historie gelöscht werden. In diesem Fall referenziert ein Datei-Objekt ein nunmehr nicht-existentes Inhalts-Objekt. Die Löschung des Namens wird zwar entdeckt, und ein Anwender sollte auf eine geeignete Weise darauf hingewiesen werden, das Datei-Objekt selbst bleibt jedoch gültig und insgesamt bleibt die Revisionssicherheit des Datenspeicher bestehen.

[0041] Bevorzugt werden sensitive Inhalts-Objekte durch eine Ende-zu-Ende Verschlüsselung vor unberechtigtem Zugriff geschützt. Eine separate Speicherung von Inhalts-Objekten ermöglicht hierbei eine nachträgliche Neuverschlüsselung.

[0042] Des Weiteren betrifft die Erfindung ein Computerprogrammprodukt. Das Computerprogrammprodukt umfasst einen maschinenlesbaren Code mit Instruktionen, welche, wenn sie auf einem Computersystem ausgeführt werden, dieses veranlassen, die Schritte eines der zuvor beschriebenen Verfahren durchzuführen.

[0043] Bevorzugt handelt es sich bei dem Computersystem um einen oder mehrere Server, welche bevorzugt in einem Rechenzentrum betrieben werden.

Kurze Beschreibung der Figuren

[0044] Ein Ausführungsbeispiel der Erfindung wird nachstehend anhand der beigefügten Abbildungen näher erläutert.

Fig. 1 zeigt ein Computersystem.

Fig. 2a zeigt den Ablauf einer Datenänderung.

Fig. 2b zeigt einen Datensatz der Datenraum-Datenbank und die Blockchain Ethereum.

Fig. 2c zeigt die Absicherung eines Wurzel-Verzeichnis.

Fig. 3 zeigt den Inhalt des CAS vor und nach der in **Fig.** 2a beschriebenen Änderung.

Fig. 4 zeigt die Absicherung eines kompletten Commit über einen Merkle-Tree.

Fig. 5 zeigt den Ablauf eines Datenexport sowie die Erstellung und Prüfung eines Beweis.

Fig. 6 zeigt eine Erweiterung des Inhalts des CAS aus **Fig.** 3 um Referenzen auf Vorgänger.

Ausführliche Beschreibung der Figuren

[0045] Im Folgenden wird zuerst anhand der **Fig.** 1 bis **Fig.** 3 ein Ausführungsbeispiel im Einklang mit der Erfindung erläutert. Dargestellt ist in **Fig.** 1 ein revisionssichere Speichersystem (100), nachfolgend auch „System“ genannt, welches zur Durchführung eines Verfahrens gemäß der Erfindung geeignet ist. Das revisionssichere Speichersystem (100), soll hierbei für eine revisionssichere Speicherung von in einer Verzeichnisstruktur organisierten Dateien genutzt werden. Die Komponenten des Systems können auf einem einzigen Computersystem (z.B. einem Server) oder verteilt (mehrere Computer-Systeme) betrieben werden.

[0046] Ein Anwender (108) greift üblicherweise über eine graphische Benutzerschnittstelle (107) auf das System zu. Im System werden Daten im CAS (101) und einer Datenraum-Datenbank (102) gespeichert. Die CAS Verwaltung (103) greift lesend und schreibend auf CAS und Datenraum-Datenbank zu. Der sog. „Notar“ (104) greift lesend auf das CAS, lesend und schreibend auf die Datenraum-Datenbank zu und greift über den Zeitstempel-Dienst-Zugangspunkt (105) auf den externen Zeitstempel-Dienst (106) zu. Im Bespiel wird als Zeitstempel-Dienst das öffentliche Ethereum-Blockchain-Computersystem genutzt und der Zeitstempel-Dienst-Zugangspunkt wäre eine sog. „Ethereum Full Node“.

[0047] Es wird angenommen, dass bereits eine Datei unter dem Pfad „/test.txt“ und eine Datei unter dem Pfad „/Rechnungen/AB123.pdf“ gespeichert wurden, dargestellt in **Fig.** 3. als Commit 0 (300).

[0048] **Fig.** 2a veranschaulicht den Ablauf einer Änderung. Der Anwender (200) initiiert den Prozess indem er über die Benutzerschnittstelle (201) eine neue Datei „test.txt“ mit geändertem Inhalt in das Wurzel-Verzeichnis hochladen will (210). Die Benutzerschnittstelle ruft (211) hierfür die CAS Verwaltung (202) auf. Die CAS Verwaltung erzeugt die benötigten Daten-Objekte, bezeichnet als „Commit 1“ (301), und speichert (212) diese im CAS (203).

[0049] **Fig.** 2b zeigt einen Datensatz der Datenraum-Datenbank, nachfolgend auch nur „Datensatz“ genannt, und eine Darstellung der Blockchain. Nach erfolgreicher Speicherung, speichert (213) die CAS Verwaltung den Hash des neuen Wurzel-Verzeichnis „r2“ (aus Commit 1) in der Datenraum-Datenbank (204), indem ein Datensatz (220) geschrieben wird, der den Hash des Wurzel-Verzeichnis und das aktuelle Datum enthält, wobei das Feld „ethereum-transaction-id“ (221) initial nicht gesetzt wird. Der Datensatz der Datenraum-Datenbank speichert die Felder (datum: „22.02.2022 00:00“, wurzel-verzeichnis-hash: „r2“). Die Änderung wurde damit gespeichert, es besteht jedoch für Commit 1 noch keine Absicherung. Dem Anwender kann eine entsprechende Erfolgsmeldung (214) angezeigt werden.

[0050] **Fig.** 3 zeigt den Inhalt des CAS vor (300) der Änderung und die zusätzlichen Daten-Objekte nach (301) der Änderung.

[0051] Dargestellt ist ein Datei-Objekt (302) mit Hash „f1“ und Feldern (Feldern name: „AB123.pdf“, datum: „20.02.2022 11:15“, inhalt: Binäre Daten). Dargestellt ist ein Datei-Objekt (303) mit Hash „f3“ und Feldern (name: „test.txt“, datum: „21.02.2022 16:35“, inhalt: „Neuer Inhalt“). Dargestellt ist ein Wurzel-Verzeichnis (304) mit Hash „r2“ und Feldern (name: "", datum: „21.02.2022 16:35“, verzeichnis-objekt-hashes: „d1“, datei-objekt-hashes: „f3“). Dargestellt ist ein Verzeichnis-Objekt (305) mit Hash „d1“ und Feldern (name: „Rechnungen“, datum: „20.02.2022 11:15“, datei-objekt-hashes: „f1“). Dargestellt ist ein Wurzel-Verzeichnis (306) mit Hash „r1“ und Feldern (name: "", datum: „20.02.2022 11:15“, verzeichnis-objekt-hashes: „d1“, datei-objekt-hashes: „f2“). Dargestellt ist ein Datei-Objekt (307) mit Hash „f2“ und Feldern (name: „test.txt“, datum: „20.02.2022 11:15“, inhalt: „Hallo Welt“).

[0052] Die genaue Ausgestaltung der Attribute des Datenmodells ist flexibel. Beispielsweise stellt Datei-Objekt (302) mit Hash „f1“ eine Datei dar, die einem Anwender unter dem Pfad „/Rechnungen/AB123.pdf“ angezeigt werden könnte, mit einem Erstellungsdatum „20.02.2022 11:15“ und einem binären Dateiinhalt. Die gezeigten Hash-Werte sind gekürzte Beispielwerte. In der Praxis werden kryptographisch sichere Hashes mit einer Länge von 256 Bit bevorzugt. Zu sehen ist ferner, die geänderte Datei „test.txt“ (303). Das neue Wurzel-Verzeichnis „r2“ (304) referenziert die geänderte Datei „test.txt“ (303) sowie das bestehende und unveränderte Verzeichnis-Objekt „d1“ (305). **Fig.** 3 veranschaulicht die erwähnten Vorteile der Nutzung des CAS in der Erfindung. Hierdurch stehen grundsätzlich für alle Daten-Objekte Hashes zu Verfügung, die alle abhängigen Daten beschreiben. Außerdem ist keine Datenduplikation erforderlich. So wird durch die Änderung an der Datei „test.txt“ hierfür ein neues Daten-Objekt

angelegt. Die Objekte für das Verzeichnis „Rechnungen“ (305) müssen jedoch nicht dupliziert werden.

[0053] Die Herstellung von Revisionssicherheit wird durch den „Notar“ (104) umgesetzt. Der Hash des aktuellen Wurzel-Verzeichnis ist hierfür wiederkehrend abzusichern. Im Beispiel wird festgelegt, dass, falls es eine Änderung im CAS gibt, eine Absicherung um 00:00 Uhr des darauf folgenden Tag zu erfolgen hat.

[0054] Fig. 2c zeigt wie eine Absicherung erstellt wird. Der Notar (231) wird jeden Tag um 00:00 Uhr ausgeführt. Er lädt (232) aus der Datenraum-Datenbank (233) den neusten Datensatz. Falls in diesem noch keine „ethereum-transaction-id“ (221) gesetzt ist, ist das aktuelle Wurzel-Verzeichnis abzusichern (andernfalls kann die Ausführung des Notar beendet werden). Der Notar schickt nachfolgend eine Anweisung (234) an den Zeitstempel-Dienst-Zugangspunkt (235), eine neue Ethereum Transaktion zu schreiben, die den Hash des aktuellen Wurzel-Verzeichnis in ihrem „Input Data“ Feld enthält und wartet bis die Transaktion geschrieben wurde. Bei Erfolg erhält der Notar die erstellte Ethereum Transaction ID als Antwort (236). Der Notar aktualisiert (237) in der Datenraum-Datenbank den Datensatz (220) des abgesicherten Wurzel-Verzeichnis, so dass dieses die entsprechende Ethereum Transaction referenziert (222). Der Notar enthält geeignete Fehlerbehandlungen z.B. um durch erneutes Probieren im Fehlerfall die erfolgreiche Absicherung des Hashes sicherzustellen. Im Beispiel würde der Notar am 22.02.2022 um 00:00 Uhr ausgeführt werden und erkennen, dass das Wurzel-Verzeichnis mit Hash „r2“ abgesichert werden muss, dessen Hash nach Ethereum schreiben, als Antwort eine Ethereum Transaction ID „s2“ erhalten und diese speichern. Somit ist ein Datensatz wie dargestellt (220) entstanden mit Feldern (datum: „22.02.2022 00:00“, wurzel-verzeichnis-hash: „r2“, ethereum-transaction-id: „s2“).

[0055] Die Datenraum-Datenbank kann bei einer trivialen Umsetzung eine simple Text-Datei sein. Es wäre auch eine andere Umsetzungsvariante denkbar, bei der auf eine separate Datenraum-Datenbank verzichtet wird und diese Informationen direkt aus einer bekannten Liste von Ethereum Transaktionen ermittelt werden.

[0056] Im Anschluss kann die Revisionssicherheit des Datenspeichers wie folgt bewiesen werden. Im Beispiel wurde der Wurzel-Hash „r2“ am 22.02.2022 00:00 Uhr über den Zeitstempel-Dienst abgesichert. Daher kann gezeigt werden, dass die geänderte Datei „test.txt“ (303), vor diesem Zeitpunkt bereits vorhanden sein musste, denn der Hash des Wurzel-Verzeichnis „r2“ (304) ist abhängig von „f3“. Der Hash „r1“ des Wurzel-Verzeichnis (306) wurde wiederum am 21.02.2022 00:00 Uhr abgesichert. Daher kann gezeigt werden, dass die ursprüngliche Datei „test.txt“ mit Hash „f2“ (307) vor diesem Zeitpunkt bereits vorhanden sein musste und „f3“ (303) noch nicht existiert hat, denn der Hash „r1“ ist abhängig von der Existenz von „f2“ und vom Fehlen von „f3“. Somit kann beweisbar die Änderung der Datei „test.txt“ dem Zeitraum 21.02.2022 00:00 Uhr bis 22.02.2022 00:00 Uhr zugeordnet werden. In der Praxis kann daher dem aufgezeichneten Datum für „f3“ (dem „21.02.2022 16:35“) vertraut werden. Da „f2“ zuvor am selben Pfad existierte, handelt es sich also um eine Änderung des Dateiinhalts von „test.txt“.

[0057] Somit ist es nicht möglich nachträglich Operationen (hier das Erstellen von „f2“) verschwinden zu lassen. Durch einen Export der Daten aus dem CAS und der Datenraum Datenbank sind die dargelegten Beweise für beliebige dritte Parteien durchführbar und damit das Ziel eines nachprüfbar revisionssicheren Datenspeichers erfüllt.

[0058] Bevorzugt ist vorgesehen, dass für ein im Speicher neu hinzugefügtes Daten-Objekt dessen Hash in einem Merkle-Tree (400) gespeichert wird, und der Hash der Spitze des Merkle-Tree (401) über den Zeitstempel-Dienst (106) abgesichert wird.

[0059] Analog zum bisher vorgestellten Vorgehen wird der Notar am 21.02.2022 um 00:00 Uhr ausgeführt. Er erkennt, dass im Datensatz für Wurzel-Verzeichnis „r1“ (306) in der Datenraum-Datenbank noch keine „ethereum-transaction-id“ gesetzt ist (221). Da keine anderen früheren Wurzel-Verzeichnisse existieren, werden alle Hashes abgesichert, von denen „r1“ abhängt. Wie dargestellt in **Fig.** 4, werden die Hashes r1, d1, f1 und f2 in einen neuen Merkle-Tree (400) eingefügt. Es wird angenommen, dass sich daraus die Spitze des Merkle-Tree mit Hash „m1“ (401) ergibt.

[0060] Der Notar schickt (234) nachfolgend eine Anweisung an den Zeitstempel-Dienst-Zugangspunkt, eine neue Ethereum Transaktion zu schreiben, die „m1“ in ihrem „Input Data“ Feld enthält und wartet bis die Transaktion geschrieben wurde. Der Notar erhält als Antwort die Ethereum Transaction ID „s1“. Analog zum bereits dargestellten Vorgehen, aktualisiert der Notar in der Datenraum-Datenbank den Datensatz des Wurzel-Verzeichnis, so dass dieses die Ethereum Transaction „s1“ referenziert. Zusätzlich wird der erstelle Merkle-Tree gespeichert, so dass ein Datensatz wie dargestellt entsteht (403) mit Feldern (datum: „21.02.2022 00:00“, wurzel-verzeichnis-hash: „r1“, merkle-tree: <binäre daten>, ethereum-transaction-id: „s1“).

[0061] Zum 22.02.2022 00:00 Uhr wiederum werden für Wurzel-Verzeichnis „r2“ die Hashes „r2“ und „f3“ in einem neuen Merkle-Tree erfasst, die Spitze

des Merkle-Tree „m2“ in der Blockchain hinterlegt und die entsprechenden Daten in einem Datensatz in der Datenraum-Datenbank gespeichert.

[0062] Fig. 5 veranschaulicht den Ablauf eines Datenexport sowie die Erstellung und Prüfung eines Beweis für die Daten-Objekte. Eine dritte Partei (500) fordert (506) über die Benutzerschnittstelle (501) einen Datenexport des Verzeichnis „Rechnungen“ so wie es zum 22.02.2022 00:00 Uhr vorlag an. Die Benutzerschnittstelle lädt (507) die Daten-Objekte „d1“ und „f1“ aus dem CAS (502). Im Anschluss fordert (508) die Benutzerschnittstelle für „d1“ einen Beweis an.

[0063] Der Notar (503) sucht (509) hierfür zuerst einen Datensatz in der Datenraum-Datenbank (504), welcher einen Merkle-Tree enthält, in dem „d1“ enthalten ist. Bevorzugt wird in allen Daten-Objekten eines Commit, die Nummer des Commit gespeichert und diese ebenfalls im Datensatz gespeichert. Hierdurch kann unmittelbar der erste Datensatz gefunden werden der einen Merkle-Tree besitzen könnte der „d1“ enthält. Es werden nun zeitlich aufsteigend, die Datensätze durchsucht. Alternativ kann eine triviale Umsetzung alle Datensätze prüfen.

[0064] Es wird somit Datensatz (403) gefunden, in dessen Merkle-Tree „d1“ enthalten ist. Im nächsten Schritt wird ein Merkle-Proof erzeugt welcher zeigt, dass sich aus „d1“ die Spitze des Merkle-Tree „m1“ errechnen lässt. Aus dem Datensatz (403) ist ferner die Ethereum Transaction ID „s1“ bekannt an der „m1“ hinterlegt (402) wurde. Der Notar liefert als Antwort (510) einen Beweis für „d1“ der den Merkle-Proof und die Ethereum Transaction ID umfasst. Die Benutzerschnittstelle liefert als Antwort (511) diesen Beweis sowie die Daten-Objekte „d1“ und „f1“ an die dritte Partei (500).

[0065] Über diesen Beweis kann eine dritte Partei die Echtheit von „d1“ und „f1“ prüfen. Zuerst wird hierfür der gegebene Merkle-Proof überprüft (512), insbesondere ob „m1“ abhängig von „d1“ ist. Im Anschluss wird die Ethereum Transaction „s1“ aus Ethereum geladen (513). Die Antwort mit der Ethereum Transaction (514) enthält in ihrem „Input Data“ Feld den Wert „m1“ und hat ein Datum an welchem sie erstellt wurde, z.B. „22.02.2022 00:05 Uhr“. Damit hat die dritte Partei bewiesen, dass „d1“ (305) und „f1“ (302) um „22.02.2022 00:05 Uhr“ existiert haben müssen, da „m1“ von „d1“ und „d1“ von „f1“ abhängig sind. Aufgrund des geringen zeitlichen Abstand zu dem in „d1“ und „f1“ gespeicherten Datum „20.02.2022 11:15“ kann man dem Datum der Daten-Objekte vertrauen. In der Praxis kann der dritten Partei eine quell-offene IT-Anwendung zur Verfügung gestellt werden, welche den beschriebenen Prozess durchführt.

[0066] Fig. 6 zeigt eine Erweiterung des Inhalts des CAS aus **Fig.** 3 um Referenzen auf vorhergehende Daten-Objekte.

[0067] Bevorzugt ist vorgesehen, dass bei einer Datenänderung im Speicher ein neues Wurzel-Objekt ein vorheriges Wurzel-Objekt referenziert. So erhält das zuvor dargestellte Wurzel-Verzeichnis mit Hash „r2“ (304) ein zusätzliches Feld (vorgänger: „r1“) wodurch es auf das vorgehende Wurzel-Verzeichnis (306) referenziert (600). Somit kann gezeigt werden, dass die durch „r1“ (306) definierte Baumregion in der Vergangenheit existiert hatte, auch wenn „r1“ nicht abgesichert worden wäre.

[0068] Bevorzugt ist vorgesehen, dass Daten-Objekte vorhergehende Daten-Objekte über deren Hash referenzieren. So erhält auch das zuvor dargestellte Datei-Objekt mit Hash „f3“ (303) ein zusätzliches Feld (vorgänger: „f2“) wodurch es auf das vorgehende Datei-Objekt (307) referenziert (601). Somit kann gezeigt werden, dass die Datei „f2“ in der Vergangenheit existiert hatte, auch wenn „r1“ (306) nicht abgesichert worden wäre.

Liste der Bezugszeichen

Fig. 1:

100	Revisionssicheres Speichersystem
101	CAS
102	Datenraum-Datenbank
103	CAS Verwaltung
104	Notar
105	Zeitstempel-Dienst-Zugangspunkt
106	Zeitstempel-Dienst
107	Benutzerschnittstelle
108	Anwender
109	Dritte Partei

Fig. 2a:

200	Anwender
201	Benutzerschnittstelle
202	CAS Verwaltung
203	CAS
204	Datenraum-Datenbank
210	Aufruf Benutzerschnittstelle
211	Aufruf CAS Verwaltung
212	Schreiben in CAS
213	Schreiben in Datenraum-Datenbank

214	Anzeige Erfolgsmeldung

Fig. 2b:

220	Ein Datensatz der Datenraum-Datenbank
221	Feld „ethereum-transaction-id“ in einem Datensatz der Datenraum-Datenbank
222	Referenz auf eine Ethereum Transaction ID

Fig. 2c:

231	Notar
232	Datensatz aus Datenraum-Datenbank laden
233	Datenraum-Datenbank
234	Aufruf Zeitstempel-Dienst-Zugangspunkt
235	Zeitstempel-Dienst-Zugangspunkt
236	Antwort mit Ethereum Transaction ID
237	Schreiben in Datenraum-Datenbank

Fig. 3:

300	Commit 0
301	Commit 1
302	Datei-Objekt mit Hash „f1“
303	Datei-Objekt mit Hash „f3“
304	Wurzel-Verzeichnis mit Hash „r2“
305	Verzeichnis-Objekt mit Hash „d1“
306	Wurzel-Verzeichnis mit Hash „r1“
307	Datei-Objekt mit Hash „f2“

Fig. 4:

400	Merkle-Tree
401	Spitze des Merkle-Tree mit Hash „m1“
402	Hash „m1“ gespeichert in einer Ethereum Transaction innerhalb eines Blocks der Blockchain
403	Ein Datensatz der Datenraum-Datenbank

Fig. 5:

500	Dritte Partei
501	Benutzerschnittstelle
502	CAS
503	Notar
504	Datenraum-Datenbank
505	Ethereum
506	Anforderung Datenexport
507	Laden von Daten-Objekten aus CAS
508	Anforderung eines Beweis
509	Suchen des Datensatz in der Datenraum-Datenbank
510	Antwort mit erstelltem Beweis
511	Antwort mit Daten-Objekten und Beweis
512	Überprüfung des Merkle-Proof
513	Laden der Ethereum Transaction
514	Antwort mit Ethereum Transaction

Fig. 6:

600	Vorgänger-Referenz auf das vorhergehende Wurzel-Verzeichnis „r1“
601	Vorgänger-Referenz auf das vorhergehende Datei-Objekt „f2“

ZITATE ENTHALTEN IN DER BESCHREIBUNG

Diese Liste der vom Anmelder aufgeführten Dokumente wurde automatisiert erzeugt und ist ausschließlich zur besseren Information des Lesers aufgenommen. Die Liste ist nicht Bestandteil der deutschen Patent- bzw. Gebrauchsmusteranmeldung. Das DPMA übernimmt keinerlei Haftung für etwaige Fehler oder Auslassungen.

Zitierte Patentliteratur

- US 4309569 [0004, 0032]
- CN 106815530 B [0006]

Patentansprüche

1. Verfahren, durchführbar mit einem Computersystem mit einem Speicher, zur Speicherung von Daten die in einer Baumstruktur organisiert und in einem Speicher eines Computersystems abgespeichert sind, insbesondere von in einer Verzeichnisstruktur organisierten Dateien, wobei der Speicher (103) einzelne Daten-Objekte über einen Hash ihres Inhalts referenziert und wiederkehrend einen Hash des aktuellen Wurzel-Objekts der Baumstruktur über einen Zeitstempel-Dienst (106) absichert.

2. Verfahren nach Anspruch 1, **dadurch gekennzeichnet**, dass der Hash über das SHA3-256 Verfahren erzeugt wird.

3. Verfahren nach einem der vorgenannten Ansprüche, **dadurch gekennzeichnet**, dass die Absicherung über den Zeitstempel-Dienst (106) in regelmäßigen zeitlichen Abständen erfolgt.

4. Verfahren nach einem der vorgenannten Ansprüche, **dadurch gekennzeichnet**, dass bei einer Datenänderung im Speicher ein neues Wurzel-Objekt ein vorheriges Wurzel-Objekt referenziert (600).

5. Verfahren nach einem der vorgenannten Ansprüche, **dadurch gekennzeichnet**, dass für ein im Speicher neu hinzugefügtes Daten-Objekt dessen Hash in einem Merkle-Tree (400) gespeichert wird, und der Hash der Spitze des Merkle-Tree (401) über den Zeitstempel-Dienst (106) abgesichert wird.

6. Verfahren nach einem der vorgenannten Ansprüche, **dadurch gekennzeichnet**, dass Daten-Objekte vorhergehende Daten-Objekte über deren Hash referenzieren (601).

7. Verfahren nach einem der vorgenannten Ansprüche, **dadurch gekennzeichnet**, dass vorbestimmte Inhalte, bevorzugt Name, Datum, Dateiinhalt, der Daten-Objekte als eigenständige Daten-Objekte über einen Hash referenziert werden.

8. Computerprogrammprodukt umfassend Instruktionen, welche, wenn diese auf einem Computersystem ausgeführt werden, das Computersystem veranlassen, die Schritte eines Verfahrens nach einem der vorhergehenden Ansprüche durchzuführen.

Es folgen 7 Seiten Zeichnungen

Anhängende Zeichnungen

Fig. 1

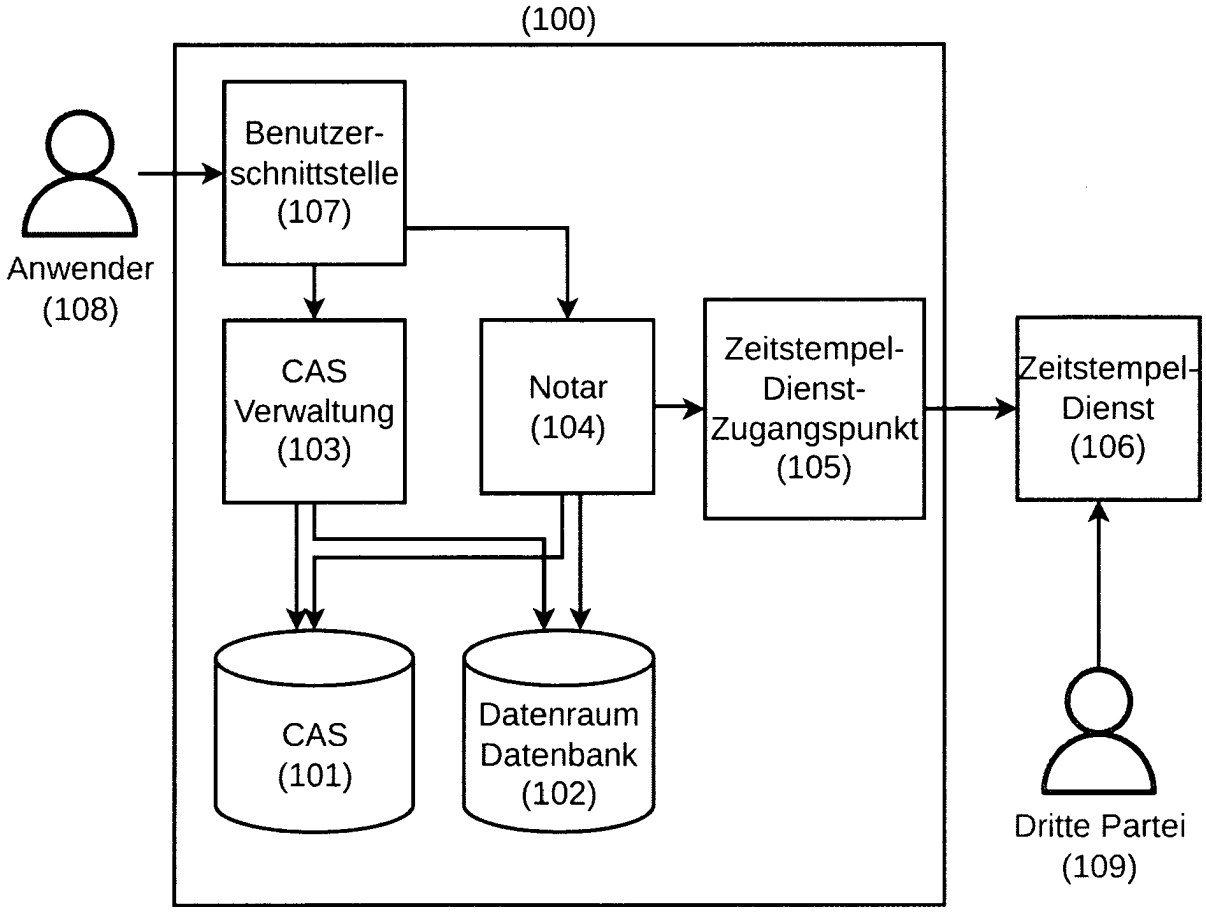


Fig. 2a

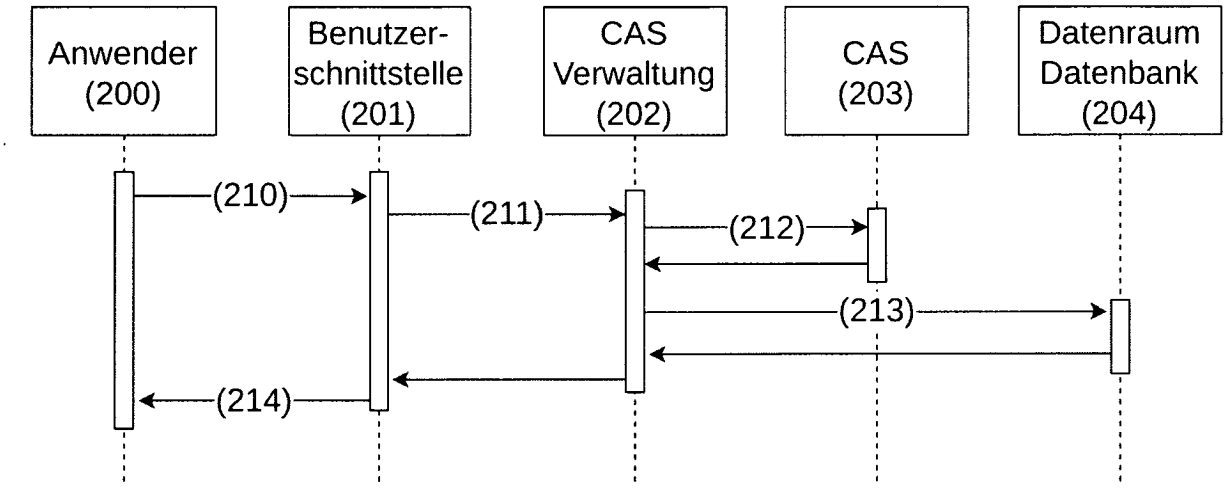

Anwender (200)
Benutzer-schnittstelle (201)
CAS Verwaltung (202)
CAS (203)
Datenraum Datenbank (204)
(210)
(211)
(212)
(213)
(214)

Fig. 2b

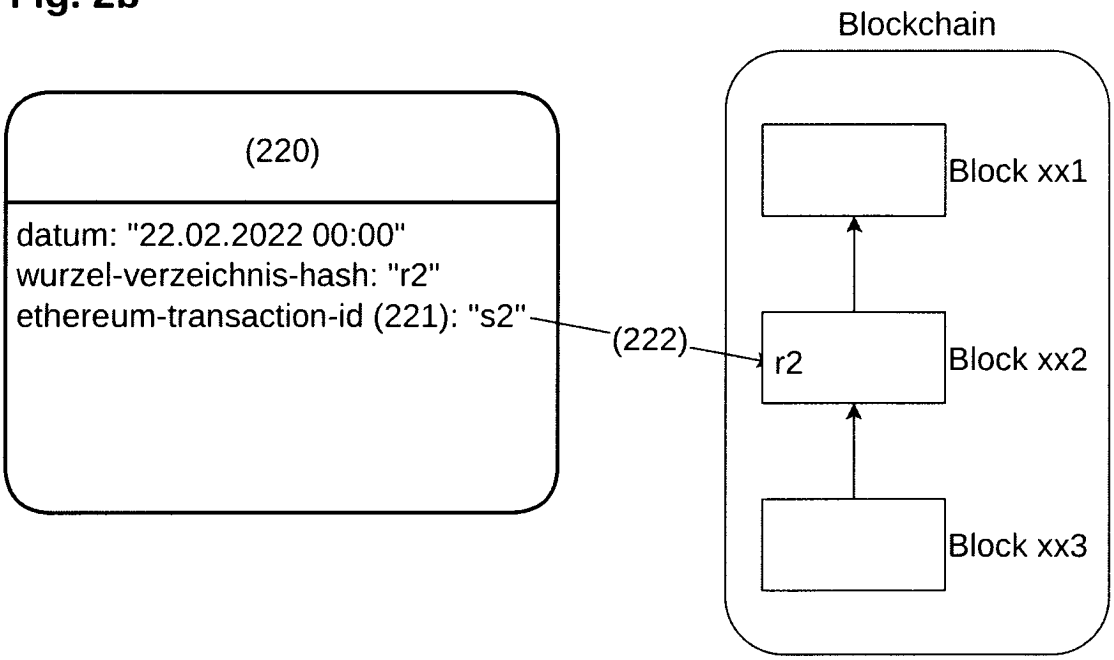

(220)
datum: "22.02.2022 00:00"
wurzel-verzeichnis-hash: "r2"
ethereum-transaction-id (221): "s2"
(222)
Blockchain
Block xx1
r2
Block xx2
Block xx3

Fig. 2c

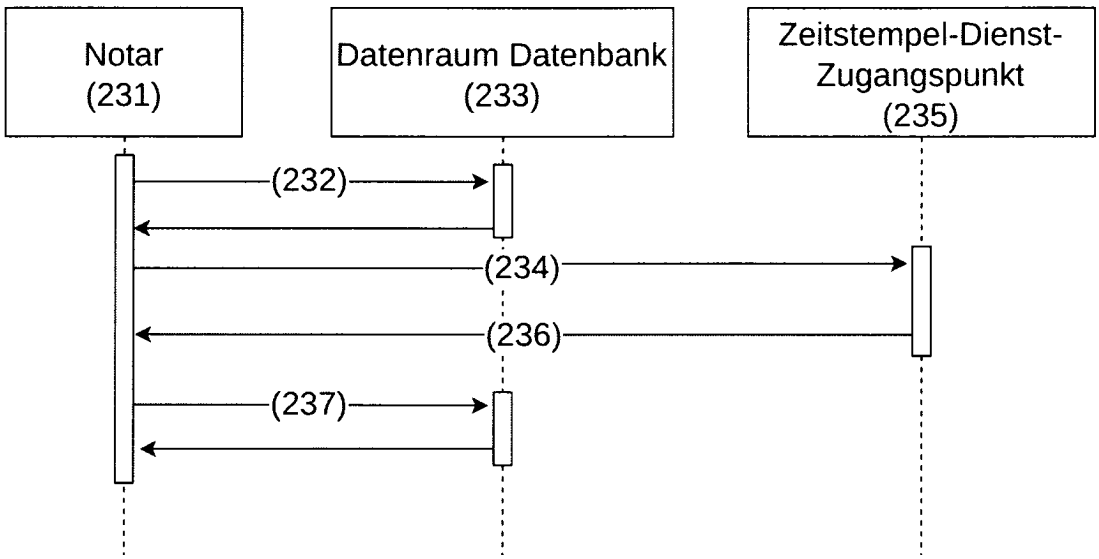

Notar
(231)
Datenraum Datenbank
(233)
Zeitstempel-Dienst-
Zugangspunkt
(235)
(232)
(234)
(236)
(237)

Fig. 3

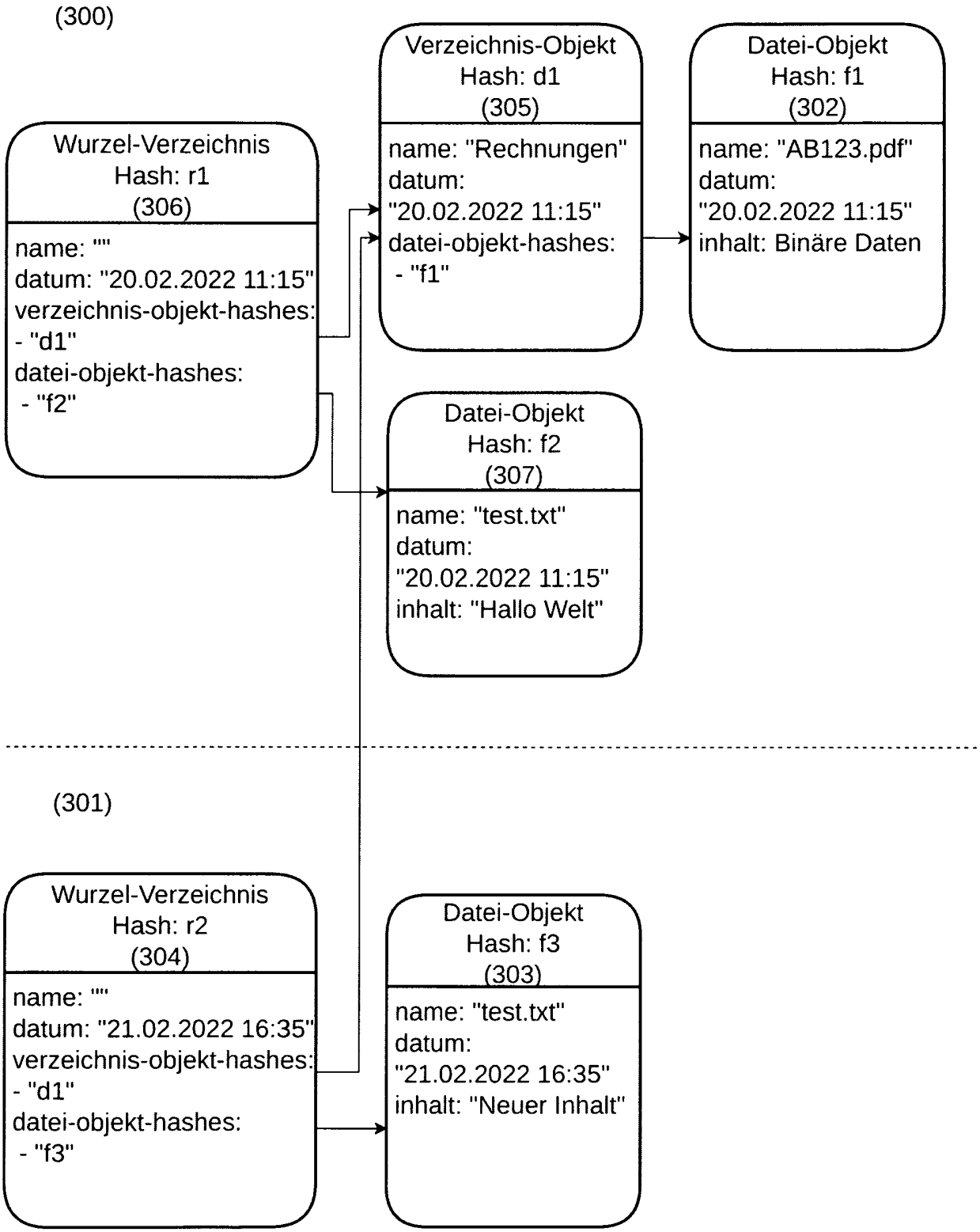

(300)
Wurzel-Verzeichnis
Hash: r1
(306)
name: ""
datum: "20.02.2022 11:15"
verzeichnis-objekt-hashes:
- "d1"
datei-objekt-hashes:
- "f2"
Verzeichnis-Objekt
Hash: d1
(305)
name: "Rechnungen"
datum:
"20.02.2022 11:15"
datei-objekt-hashes:
- "f1"
Datei-Objekt
Hash: f1
(302)
name: "AB123.pdf"
datum:
"20.02.2022 11:15"
inhalt: Binäre Daten
Datei-Objekt
Hash: f2
(307)
name: "test.txt"
datum:
"20.02.2022 11:15"
inhalt: "Hallo Welt"
(301)
Wurzel-Verzeichnis
Hash: r2
(304)
name: ""
datum: "21.02.2022 16:35"
verzeichnis-objekt-hashes:
- "d1"
datei-objekt-hashes:
- "f3"
Datei-Objekt
Hash: f3
(303)
name: "test.txt"
datum:
"21.02.2022 16:35"
inhalt: "Neuer Inhalt"

Fig. 4

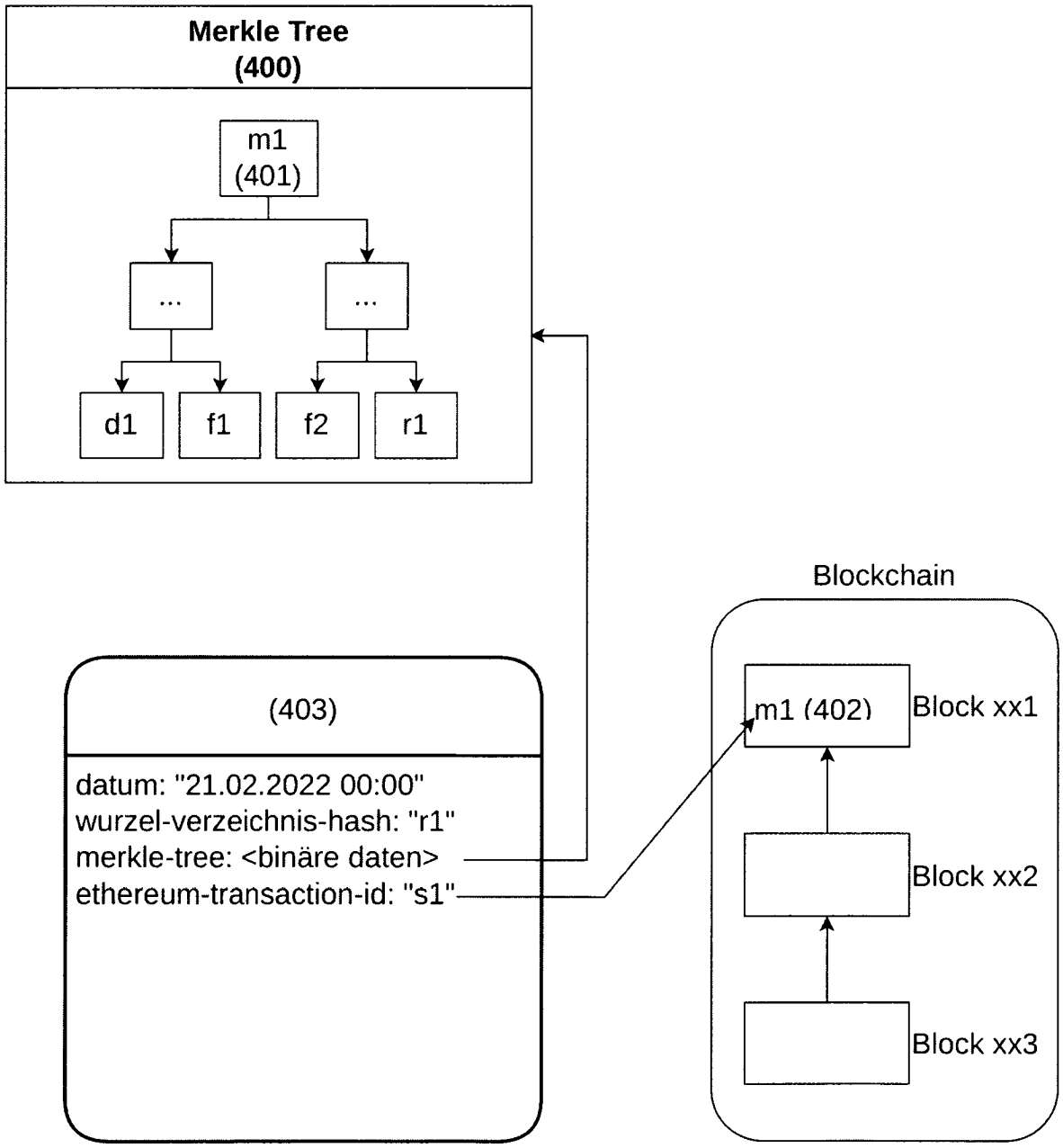

Merkle Tree
(400)
m1
(401)
...
...
d1
f1
f2
r1
Blockchain
(403)
datum: "21.02.2022 00:00"
wurzel-verzeichnis-hash: "r1"
merkle-tree: <binäre daten>
ethereum-transaction-id: "s1"
m1 (402)
Block xx1
Block xx2
Block xx3

Fig. 5

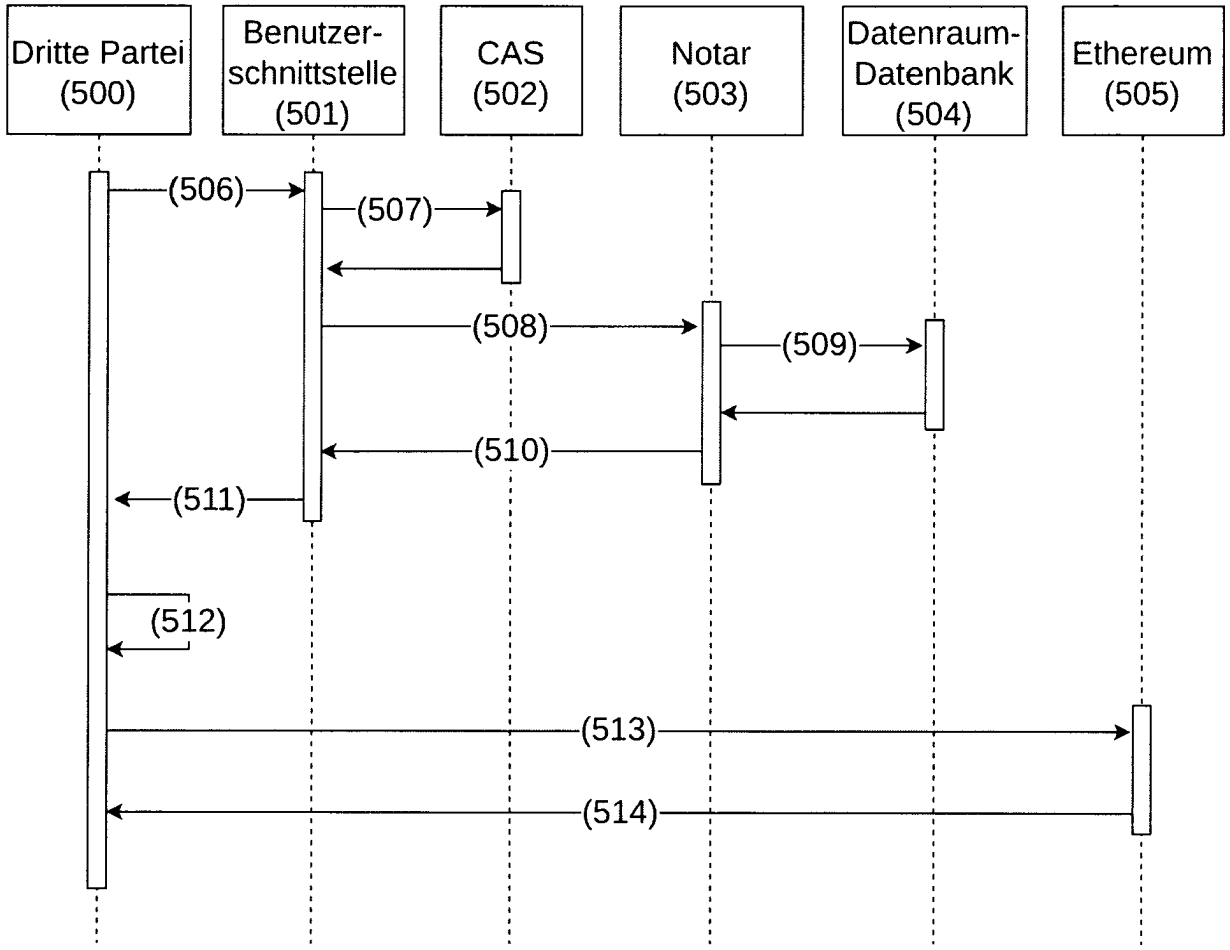

Dritte Partei (500)
Benutzer-schnittstelle (501)
CAS (502)
Notar (503)
Datenraum-Datenbank (504)
Ethereum (505)
(506)
(507)
(508)
(509)
(510)
(511)
(512)
(513)
(514)

Fig. 6