Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

FAKULTÄT
FÜR MATHEMATIK, INFORMATIK
UND NATURWISSENSCHAFTEN

Bachelorarbeit

Blockchainbasiertes Zugriffsrechte Management für sensible Daten in dezentralen Cloud-Speichersystemen

Artem Eger

invy009@informatik.uni-hamburg.de
Studiengang Software-System-Entwicklung
Matr.-Nr. 6860250
Fachsemester 6

Erstgutachter: Wolf Posdorfer
Zweitgutachter: Dr. Dirk Bade

Abgabe: 31.01.2019

Inhaltsverzeichnis

1 Einleitung

Die Geschäftsbelastung bei Gerichten, Staatsanwaltschaften und damit auch bei beteiligten Anwälten, ist bereits seit 1999 konstant auf einem hohen Niveau. So gab es beispielsweise im Jahr 2016 5,2 Millionen neuer Verfahren für Staatsanwaltschaften an deutschen Landgerichten [Jus16]. Die durch den Dokumentationsaufwand entstehende Flut von sensiblen Briefen, Akten und Dateien pro Verfahren, ist folglich um ein vielfaches höher als die Anzahl der Verfahren selbst. Um die Digitalisierung in diesem Sektor vorantreiben zu können, benötigen die Beteiligten eine effiziente Lösung für die Persistenz und den Austausch von Dateien mit hohen Datenschutzanforderungen.

Eine digitale Lösung für das Problem der anfallenden Aktenflut zu finden, hat neben dem ressourcenschonenden Aspekt, viele andere Vorteile für den Rechtssektor. Juristen haben beispielsweise, bis zu dem Zeitpunkt an dem digitale Versionen von Akten und Briefen ausgetauscht werden können, keine Möglichkeit die aktuellen Fortschritte im Bereich sprach basierter KI zu nutzen. Eine amerikanische KI-Plattform hat in einer Studie gezeigt, dass menschliche Anwälte in bestimmten Aspekten ihrer Arbeit durch einen Algorithmus ersetzt werden können [Law17]. Solche Projekte könnten unter anderem den Arbeitsaufwand von Anwälten und Richtern drastisch verringern, erfordern jedoch zu aller erst eine stabile Infrastruktur über die Dokumente digital und sicher ausgetauscht werden können. Da davon ausgegangen werden muss, dass Personen aus dem juristischen Umfeld nicht zwingend IT-affin sind, müssen wichtige Kernkomponenten wie Kryptographie bereits per Design in diese Infrastruktur integriert werden.

1.1 Problembeschreibung

Aufgrund einer fehlenden Softwarelösung müssen Anwälte noch immer miteinander über den Postweg oder ein spezielles Anwaltspostfach im Gerichtsgebäude kommunizieren. Akten werden ebenfalls mit der Post zugestellt und von jedem Beteiligten für seinen persönlichen Gebrauch nochmals vervielfältigt. Dieser Arbeitsvorgang impliziert ein hohes Maß an Papierverschwendung und langsamen Kommunikationswegen. Durch den Mangel an digitaler Infrastruktur, können juristische Softwarelösungen keine Tools zur direkten Arbeit mit Textinhalten von Dokumenten bereitstellen. Obwohl nahezu jedes juristische Dokument sensible Informationen enthält und entsprechend behandelt werden sollte, gibt es hierfür bisher keinerlei kryptographische Anforderungen. Darüber hinaus fehlt bisher ein sicherer Prozess, mit dem sensible digitale Dokumente ausgetauscht wer-

den können. Zwar gab es vereinzelt Versuche, über das elektronischem Anwaltspostfach BeA [Bun18], eine sichere Infrastruktur für den Austausch bereitzustellen, jedoch musste diese Lösung kurz vor ihrer Verwendung aufgrund schwerer Sicherheitslücken abgeschaltet werden [Gol18b]. Nach wie vor scheint das BeA der Bundesanwaltskammer nicht adäquat genug implementiert, als dass es bereit wäre produktiv von Anwälten in Deutschland eingesetzt zu werden [Gol18a].

1.2 Wissenschaftlicher Beitrag

Diese Arbeit zielt darauf ab ein Gesamtkonzept zu erarbeiten, welches eine mögliche Infrastruktur zum transparenten und sicheren Austausch von Daten modelliert. Dieses Konzept soll sich, unabhängig vom bisher beschriebenen juristischen Sektor, mit der Problemstellung auseinandersetzen und damit auch in anderen Bereichen mit ähnlichen Anforderungen einsetzbar sein.

Hierfür werden aktuelle Technologien aus den Bereichen Kryptographie, Cloudspeicher-Systeme und verteilte Datenbanken, welche für die Lösung der Problemstellung relevant sein könnten, vorgestellt und anhand ihrer Eignung evaluiert. Außerdem werden existierende Lösungskonzepte ähnlicher Problemstellungen diskutiert und in die Entwicklung des hier vorgestellten Ansatzes mit einbezogen.

Das hier vorgestellte Modell soll folgende Kriterien erfüllen:

1. Eindeutige Identifikation von Teilnehmern
2. Vertrauliche Persistierung von Daten
3. Transparenter und vertraulicher Austausch von Daten
4. Redundante Persistierung von Daten

1.3 Abgrenzung

In der Problembeschreibung wird unter anderem die fehlende digitale Verfügbarkeit von Dokumenten bemängelt. Da diese Verfügbarkeit nur relevant sein kann, wenn auch eine entsprechende Infrastruktur vorhanden ist um diese Dokumente effizient und sicher auszutauschen, liegt der Fokus dieser Arbeit vor allem auf der Konzipierung dieser. Für die Entwicklung des Konzeptes ist es zunächst irrelevant ob die zu übertragenden Daten „gut“ oder „schlecht“ digitalisiert wurden bzw. in welcher digitalen Form sie vorliegen.

1.4 Struktur der Arbeit

Die gesamte Arbeit lässt sich folgendermaßen strukturieren:

- Im zweiten Kapitel werden Grundbegriffe und Technologien vorgestellt die für das unmittelbare Verständnis der Arbeit benötigt werden. Es wird auf eine spezielle Form einer verteilten Datenbank, Content Delivery Networks und bekannte kryptographische Konzepte eingegangen.
- Im dritten Kapitel werden Verwandte Arbeiten vorgestellt, welche die erwähnten Technologien bereits in eigenen Konzepten oder Produkten implementiert haben.
- Im vierten Kapitel wird das Gesamtkonzept erläutert, welches die aufgestellten Kriterien erfüllen soll. Dabei werden Überlegungen und Abläufe näher erläutert.
- Im fünften Kapitel wird eine mögliche Architektur des Konzeptes erarbeitet und dafür wichtige Kernkomponenten und Prozesse modelliert.
- Im letzten Kapitel wird die Arbeit nochmals kurz zusammengefasst und ein Fazit aus dem Erarbeiteten gezogen. Des weiteren wird ein Ausblick auf mögliche Verbesserungen gegeben.

2 Grundlagen

Um der Konzipierung dieser Arbeit besser folgen zu können, werden in diesem Kapitel einführend relevante Technologien aus den Bereichen Kryptographie und verteilte Systeme vorgestellt. Sollte es hiervon unterschiedliche technologische Ausprägungen dieser geben, werden verschiedene Möglichkeiten vorgestellt um diese dann, abschließend im Fazit, für die Verwendung im Konzept gegenüberstellen zu können.

2.1 Blockchain

Die Blockchain ist ein Konzept für eine verteilte Datenbank welche genau aus zwei Arten von Datensätzen besteht, nämlich Blöcken und Transaktionen [Car17]. Sie fungiert als implizite Vertrauensinstanz in einem Netzwerk von Benutzern die sich gegenseitig nicht vertrauen können. Außerdem liefert sie transparent einen, für alle gleichberechtigten Teilnehmer, konsistenten und aktuellen Zustand der Datenbank [CV17]. Ein Block besteht aus einer Menge von Transaktionen und einem digitalem Fingerabdruck seines Vorgängerblocks [Car17]. Dieser digitale Fingerabdruck wird auch Hash genannt weil er durch eine Hashfunktion erzeugt wird. Eine Hashfunktion ist eine nicht injektive Abbildung einer großen Eingabemenge auf eine kleinere Zielmenge und bietet eine einmalige Identifikation dieser [Spi11, p. 33].

Da die Blockchain ein spezielles Konzept für eine verteilte Datenbank ist, benötigt sie implizit eine Persistenz-Komponente mit der valide Transaktionen in Blöcken abgespeichert werden können [But14, p. 8]. Da der Blockchain eine verteilte Architektur zugrunde liegt wird auch eine P2P-Komponente für Kommunikation der Teilnehmer im Netzwerk benötigt [But14, p. 34]. Jede Transaktion muss vom Netzwerk deterministisch auf Semantik und Gültigkeit hin validiert werden können. Dies kann beispielsweise mithilfe eines netzwerkinternen Zustandsautomaten geschehen [But14, pp. 4]. Außerdem muss sich das Netzwerk einig darüber sein können, was in jedem Moment der letzte valide Zustand der verteilten Datenbank ist. Diese Funktionalität kann durch verschiedene Konsensprotokolle realisiert werden [CV17]. Um einen „Konsens“ in einem verteilten System herstellen zu können werden nach Schneider [Sch90] zwei Hauptkomponenten benötigt. Erstens der bereits erwähnte deterministischer Automat, welcher die Logik seines Services bereitstellt und für alle Teilnehmer repliziert und zweitens ein Konsens Protokoll, welches Anfragen an das Netzwerk so unter den Knoten verbreitet, dass das gesamte System diese in einer einheitlichen Reihenfolge bearbeiten kann.

2.2 Proof-of-Work Konsens

Die Proof-of-Work Konsensbildung in einer Blockchain basiert auf der Aufwendung von Rechenleistung um einen gültigen Block innerhalb des Netzwerkes vorschlagen zu können. Dabei muss ein neuer Block bestimmte Anforderungen erfüllen um als gültig anerkannt zu werden. Das Problem welches die Gültigkeit eines Blocks definiert, kann nicht auf eine effiziente Weise gelöst werden, sondern muss erraten werden. Dieser Ratevorgang wird im Proof-of-Work Konsens auch Mining genannt. Die Schwierigkeit des zu lösenden Problems skaliert hierbei gemeinsam mit der Rechenleistung die im gesamten Netzwerk zur Verfügung steht. Der Grundgedanke bei böswilligen Teilnehmern ist, dass einzelne diese die benötigte Rechenkapazität für einen erfolgreichen Angriff auf das Netzwerk nicht aufbringen können, bzw. dass es für sie (finanziell) unlukrativ ist viel Rechenleistung aufzuwenden um eine Manipulation vorzunehmen [Nak08].

2.2.1 Bitcoin

Das Bitcoin Netzwerk benutzt einen Proof-of-Work Algorithmus für die Konsensbildung. Das bereits erwähnte Problem durch das ein gültiger Block bei Bitcoin definiert wird, sind die führenden Nullen eines SHA256 Blockhashes. Das bedeutet konkret, dass ein Validator Transaktionen aus dem Mempool in einen Block packt und darüber einen Hash bildet. Damit der Block angehängt werden kann muss dieser Hash beispielsweise 10 führende Nullen aufweisen. Also wird ein redundanter Parameter eingeführt, die sogenannte *nonce*, welcher solange inkrementiert wird bis durch Zufall der resultierende Hash die benötigten Anforderungen erfüllt. Sollten durch parallel laufende Vorgänge bei Bitcoin mehrere Chains entstehen, ist immer die gültig, welche am längsten ist und somit die meiste Rechenpower investiert hat. Das Bitcoin Netzwerk reagiert dynamisch auf die steigende Rechenleistung innerhalb des Netzwerkes. Die Schwierigkeit des zu lösenden Problems orientiert sich dabei an dem Median der gebildeten Blöcke pro Stunde. Wenn vom Netzwerk mehr Blöcke gebildet werden steigt die Schwierigkeit [Nak08].

2.3 Proof-of-Stake Konsens

Der Proof-of-Stake Konsens ist eine ressourcenschonende Variante der Konsensbildung, welche ohne die implizierte Verschwendung von Rechenpower auskommt, indem Teilnehmer mit einer Art Anteil (*Stake*) im Netzwerk abstimmen. Die Argumentation, weswegen Proof-of-Stake funktionieren soll, ist, dass Teilnehmer mit vielen Anteilen auch an der Funktionalität und Gesundheit des Netzwerkes interessiert sind [IB17]. Je mehr Anteile ein Teilnehmer besitzt, desto höher ist seine Stimmkraft. Wenn ein Teilnehmer sich dafür entscheidet, das Netzwerk böswillig zu manipulieren benötigt er einen sehr großen Stake oder andere Teilnehmer die sich an dem Angriff beteiligen. Ein solches Szenario würde dem Profitgedanken der Angreifer widersprechen, da sie nur dem Netzwerk

schaden, an dem sie selbst zu überwiegenden Teilen beteiligt sind [IB17]. Um einen Angriff auf das Netzwerk zusätzlich unattraktiv zu machen, sind bei Proof-of-Stake Verfahren auch Bestrafungen einzelner Teilnehmer, wie das Einbehalten ihres Stakes, bei einer Falschabstimmung möglich [IB17, pp. 10]. Im Gegensatz zu Proof-of-Work können Netzwerkteilnehmer einfach Blöcke bilden und diese im Netzwerk vorschlagen, ohne gültige Blöcke mit viel Rechenaufwand erraten zu müssen. Dabei haften sie für ihren Vorschlag mit ihren Anteilen am Netzwerk [IB17].

2.3.1 Tendermint

Tendermint ist eine Blockchain Software Engine die nach dem gleichnamigen Konsens Protokoll benannt ist [Ten18]. Sie stellt alle oben erwähnten Komponenten einer Blockchain bereit. Die zwei Hauptkomponenten nach Schneider[Sch90] sind einmal der *Tendermint Core*, welcher als Konsens Protokoll fungiert und das *generic application interface* welches den Entwicklern die Implementierung eines deterministischen Automaten zur Validierung einer eigens definierten Semantik ermöglicht [Ten18]. Tendermint erlaubt eine programmiersprachenagnostische Implementierung des Zustandsautomaten. Viele moderne Programmiersprachen enthalten jedoch nicht deterministische Funktionalität. Deshalb müssen vor allem Entwickler darauf achten wirklich deterministische Anwendungen zu schreiben und nicht deterministische Funktionalität verschiedener Programmiersprachen nicht zu verwenden.

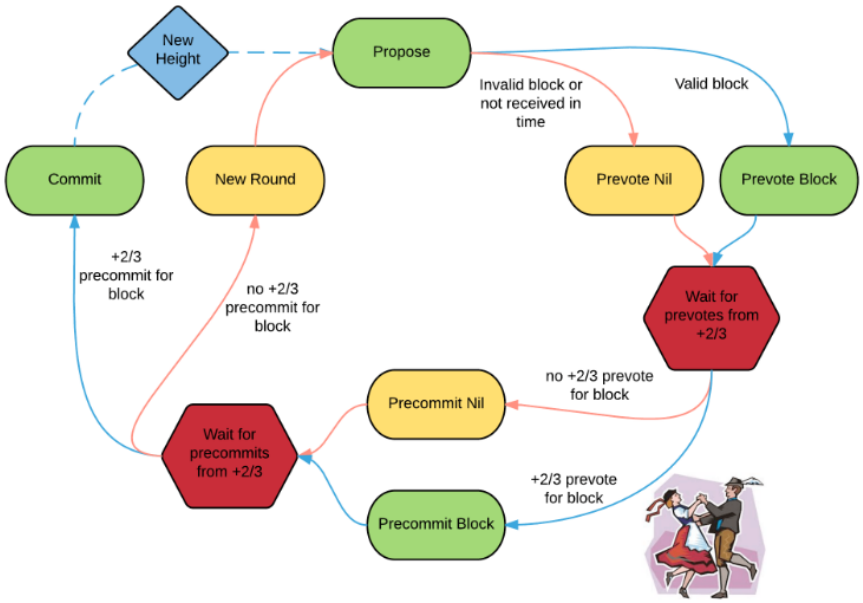

Abbildung 2.1: Tendermint Consensus [Ten18]

Tendermint Core ist ein „byzantine fault tolerance“ (BFT) Proof-of-Stake Protokoll und

kann bis zu 1/3 byzantinischer Knoten im Netzwerk tolerieren. Beteiligte, die am Protokoll teilnehmen und über Stimmkraft verfügen, welche durch ihren Stake repräsentiert wird, werden Validatoren genannt. Diese Validatoren befüllen Blöcke mit Transaktionen aus dem gemeinsamen Mempool heraus und stimmen über die Persistierung dieser Blöcke ab. Die Abstimmung erfolgt zweigeteilt (2-phase-commit) in einem *pre-vote* und einem *pre-commit*. Dies bedeutet konkret, dass bevor über das Anhängen eines neuen Blocks abgestimmt werden darf, eine erfolgreiche Vorabstimmung abgehalten werden muss (*polka*). Die Persistierung erfolgt in einer Kette mit einem Block pro Kettenglied. Die Position des Kettengliedes in der Gesamtkette wird als Höhe bezeichnet. Sollte ein vorgeschlagener Block von der Menge der Validatoren abgelehnt werden, beginnt eine neue Abstimmungsrunde in der ein anderer Validator einen Block für die gleiche Höhe vorschlägt. Ein Block wird erst dann zur Persistierung freigegeben, wenn mehr als 2/3 der Validatoren in einer Abstimmungsrunde nach einer *polka* den Block vorgeschlagen haben [Ten18].

2.4 Content Delivery Network

Um Daten für Benutzer einer verteilten Applikation bereitzustellen, werden Endpunkte im WAN benötigt die allen Beteiligten gleichermaßen zugänglich sind. Da die naive Lösung Server in einem einzelnen großen Datacenter zu betreiben global schlecht skaliert, werden so genannte *Content Delivery Networks* (CDN) benutzt um Daten gleichermaßen auf vielen Servern global zu verteilen, um diese von vielen geographischen Orten aus gleichermaßen zugänglich machen zu können [MSD+06]. Jeder Service im Internet, seien es Streaming Plattformen wie YouTube oder Cloud Services wie Dropbox, der Daten global an Benutzer bereitstellen möchte verwendet solche CDNs, um die Antwortzeiten und den Durchsatz seiner Applikationen entsprechend optimieren und skalieren zu können. Außerdem wird dadurch ein Single-Point-of-Failure im Bezug auf Verlust und Verfügbarkeit der Daten eliminiert.

2.4.1 Cloudflare

Cloudflare ist ein kommerzieller CDN Anbieter, welcher zusätzlich zur Bereitstellung der Daten, weitere Leistungen zum Schutz und zur Performancesteigerung in seinem Produkt integriert hat. Die Server sind sehr gut global verteilt und ein spezieller Routing Mechanismus ist implementiert, sodass globale Anfragen automatisch an das nächste verfügbare Datacenter weitergeleitet werden und dadurch die Antwortzeiten für Benutzer sinken. Außerdem bietet Cloudflare seinen Kunden Schutz vor gängigen webbasierten Attacken wie Distibuted-Denial-of-Service (DDos) bei der Nutzung ihres Produkts. Ein weiteres Feature von Cloudflare ist ihr patentiertes Cache Beschleunigungsprotokoll mit dem Namen *Railgun*. Dieses sorgt für intelligentes caching, das selbst bei nicht richtig konfigurierten Seiten oder Services mit dynamischem Inhalt, eine Performancestei-

gerung herbeiführen kann. Effizient ist dieses vor allem, weil es auf Byte Ebene arbeitet und damit wirklich in der Lage ist nur die Daten zu aktualisieren, bei denen eine effektive Änderung festgestellt werden konnte. Das *Railgun* Protokoll besteht aus zwei Hauptkomponenten, einem „Sender“ und einem „Listener“. Der „Listener“ ist eine Softwarekomponente welche beim Kunden mit installiert wird und den Überblick über Datenänderungen auf Kundenseite behalten soll. Der „Sender“ läuft auf allen Cloudflare-Servern und ist ständig in Kontakt mit den „Listenern“. Bei jeder Anfrage die Latenzzeiten mit sich führen würde, versucht die „Sender“ Komponente Optimierungen durch Routing und Kompression herbeizuführen. Durch *Railgun*, so Cloudflare, konnten bei bestimmten Kunden Performanceoptimierungen von bis zu 700% erreicht werden [Inc17].

2.4.2 IPFS

Das *Inter Planetary File System* (IPFS) ist ein dezentrales CDN Dateisystem, dessen Ziel es ist eine global verfügbare Festplatte bereitzustellen mit der sich alle Geräte über das Internet verbinden können. Das IPFS Projekt wird opensource betrieben und von keinem bestimmten Anbieter alleine angeboten. Es bietet seinen Nutzern ein Daten adressiertes Persistenzmodell und Daten adressierte Hyperlinks. Das bedeutet, dass jeder Datensatz der im IPFS abgelegt wurde über HTTP und dem zugehörigen Hash des Datensatzes global abgerufen werden kann. Neue Daten werden in Blöcken, wie in der Blockchain hintereinander gehängt, wodurch das gesamte System als ein einziger gerichteter azyklischer Graph gesehen werden kann. Ähnlich wie bei der Blockchain sind dadurch Änderungen an Daten im Nachhinein nicht möglich. Wird eine neue Version eines Datensatzes benötigt, so muss dieser in seiner Gesamtheit neu hochgeladen werden und bekommt dadurch auch einen neuen Hash zur eindeutigen Identifikation [Ben18]. Das IPFS Protokoll vereint viele Ideen anderer Peer-to-Peer Anwendungen wie Git oder BitTorrent und ist unterteilt in verschiedene Untermodule, die jeweils für eine bestimmte Funktionalität verantwortlich sind [Ben18]:

- Identität - Verwaltet die Erstellung neuer Knoten und deren Verifizierung. Jeder Knoten im IPFS ist durch eine *NodeId*, welche von einer Hashfunktion aus seinem öffentlichen Schlüssel erzeugt wird, identifizierbar. Wenn ein Knoten sich mit einem anderen verbindet, tauschen diese ihre öffentlichen Schlüssel aus und überprüfen ob sie mit der Anwendung der selben Hashfunktion die *NodeId* erzeugen können. Ist dies nicht der Fall, wird die Verbindung abgebrochen [Ben18].

- Netzwerk - Verwaltet die Verbindungen einzelner Knoten untereinander mithilfe verschiedener Netzwerkprotokolle. Das IPFS ist dabei nicht beschränkt auf die Verfügbarkeit des IP-Protokolls. Dadurch ist es möglich IPFS-Knoten auch in einem internen Overlay-Netz über dem globalen Internet-Netz zu betreiben. Das IPFS speichert dafür die unterschiedlichen Adressen, gekapselt mit Meta-Informationen zu verwendeten Protokollen in *multiaddr* [Mul17] formatierten Bytestrings [Ben18].

- Routing - Verwaltet Informationen um einzelne Knoten im Netzwerk gezielt finden zu können. Das IPFS benutzt dafür die Verwaltung der Metadaten Distributed-Sloppy-Hash-Tables (DSHT), welche eine optimierte Form normaler Distributed-Hash-Tables sind. Diese werden benötigt um möglichst effizient Verbindungsinformationen der Knoten untereinander speichern und abzufragen zu können [Ben18].
- Austausch - Sorgt für die effiziente Verteilung neuer Blöcke unter den Knoten. Dafür wird im IPFS das von BitTorrent inspirierte Protokoll *Bitswap* verwendet. Bitswap verhält sich wie ein persistenter Marktplatz, an dem sich Knoten jeden beliebigen Block laden können den sie benötigen [Ben18].
- Objekte - Ein globaler gerichteter azyklischer Graph der alle Identifikationshashes und deren Hierarchie enthält. Durch diesen werden die Features wie Adressierung über einen eindeutigen Hash, Integrität der Daten und die Vermeidung von Duplikaten, ermöglicht [Ben18].
- Daten - Ein Dateisystem zur Persistierung ähnlich dem von Git. Die Daten selbst werden hierbei in binärer Form abgespeichert. Sollte es Daten geben, die aufgrund ihrer Größe oder ähnlichem, aufgespalten werden müssen, bietet das IPFS Listen und Bäume um die ursprüngliche Struktur dieser Daten erhalten zu können [Ben18].
- Benennung - Ein anpassbares Bennenungssystem welches sich selbst zertifiziert. Dadurch wird es Benutzern ermöglicht verschiedene Versionen spezifischer Daten unter einem fest definierten Pfad erreichen zu können, ohne selbst die Hashwerte der sich ändernden Objekte verwalten zu müssen [Ben18].

2.5 Kryptographie

Bei kryptographischen Verfahren wird vor allem zwischen symmetrischen und asymmetrischen Verfahren unterschieden. Das symmetrische Verschlüsselungsverfahren verwendet den gleichen Schlüssel zum ver- und entschlüsseln der Daten [Spi11, pp. 16]. Heutzutage werden diese Verfahren vor allem für große Datenmengen benutzt, weil sie besonders wenig Rechenaufwand benötigen und dementsprechend schnell sind [Spi11, p. 31]. Sie gelten als sicher vor Angriffen wenn eine ausreichend lange Schlüssellänge gewählt wird, mindestens aber 112 Bit [Spi11, p. 43]. Zu dem verschlüsselten Datensatz ist es bei einem symmetrischen Verfahren erforderlich dem Empfänger den zugehörigen Schlüssel in irgendeiner Weise zugänglich zu machen. Dafür werden heutzutage asymmetrische Verfahren verwendet [Spi11, p. 43]. Einer der bekanntesten modernen Vertreter symmetrischer Verfahren ist der *DES (Data Encryption Standard)*, was jedoch so nicht mehr empfohlen werden kann, da ein 56-Bit DES Schlüssel bereits im Jahre 1998 innerhalb von 56 Stunden berechnet werden konnte. Um diesen Algorithmus dennoch weiter verwenden zu können wurde der Triple-DES (3DES) eingeführt, der drei DES-Verschlüsselungen kaskadiert [Spi11, p. 57]. Dadurch erhöht sich der Aufwand bei einem

112 Bit 3DES Schlüssel für einen erfolgreichen Angriff um den Faktor 2^{56} [Spi11, p. 58]. Dieser Algorithmus eignet sich Aufgrund seiner hohen Geschwindigkeit gut für große Datenmengen und Multimediaströme [Spi11, p. 58].

Bei asymmetrischen Verschlüsselungsverfahren werden Schlüsselpaare verwendet. Es gibt einen öffentlichen Schlüssel, der zum Verschlüsseln der Daten benutzt wird und einen privaten Schlüssel, welcher nur dem Besitzer des zugehörigen öffentlichen Schlüssels bekannt sein sollte. Mit diesem privaten Schlüssel können Daten entschlüsselt werden, die mit dem zugehörigen öffentlichen Schlüssel verschlüsselt wurden [Spi11, p. 109]. Nach dem ein Datensatz mit einem öffentlichen Schlüssel verschlüsselt wurde, kann dieser nicht mit dem selbigen entschlüsselt werden. Der Vorteil gegenüber der symmetrischen Verschlüsselung ist, dass nun der Schlüssel selbst nicht mehr zugänglich gemacht werden muss, da der private Schlüssel immer nur beim Empfänger liegt.

Ein asymmetrischen Verfahren ist nicht so performant wie ein symmetrisches. Der RSA ist im Vergleich zum DES etwa um den Faktor 1000 langsamer [Spi11, p. 123]. Deshalb werden oftmals beide Verfahren gemischt, wobei der Datensatz selbst mit einer symmetrischen Chiffre verschlüsselt wird und der resultierende symmetrische Schlüssel dann asymmetrisch [Spi11, p. 123]. Die bekanntesten und verbreitetsten asymmetrischen Verschlüsselungsverfahren sind das *RSA (Rivest, Shamir, Adelman)* und das *Elliptic Curve Cryptography* Verfahren [Spi11, s. 109].

2.5.1 RSA

Das RSA Verfahren ist, wie bereits erwähnt, ein häufig eingesetzter Algorithmus zur asymmetrischen Verschlüsselung und löst folgendes Problem:

Sender A möchte Empfänger B eine geheime Nachricht über einen unsicheren öffentlichen Kanal zukommen lassen. Dazu werden folgende Schritte benötigt [RSA83]:

1. Empfänger B
   - wählt zwei zufällige und ausreichend große Primzahlen p und q
   - berechnet die Variable N welche das Produkt aus p und q darstellt
   - berechnet die Eulersche Phi-Funktion mit $\varphi(N) = (p-1)\,(q-1)$
   - wählt eine zufällige natürliche Zahl e mit $1 < e < \varphi(N)$, sodass e und $\varphi(N)$ teilerfremd sind.
   - berechnet die Variable d mit $d \equiv e^{-1}\ mod(\varphi(N))$ mit dem euklidischen Algorithmus

   Der Empfänger B veröffentlicht die Variablen N und e als öffentlichen Schlüssel. d, p und q bleiben geheim.

2. Sender A
    - besorgt den öffentlichen Schlüssel des Empfängers
    - verschlüsselt die Nachricht m zu c mit $c \equiv m^e \; mod(N)$
    - sendet c an den Empfänger
3. Entschlüsselung durch den Empfänger
    - Die empfangene Nachricht c kann nur von B entschlüsselt werden durch: $c^d \equiv m^{ed} \equiv m^{1+j\varphi(N)} \equiv m \; mod(N)$ wobei $j \in \mathbb{N}$ sodass $ed = 1 + j\varphi(N)$

Dieses Verfahren beruht und funktioniert Aufgrund folgender mathematischen Tatsachen

- Die Berechnung der Funktion $m \mapsto m^e \; mod(N)$ ist einfach
- Die Invertierung $c \mapsto c^{\frac{1}{e}} \; mod(N)$ mit $c \in \mathbb{N}$ und $c \in 1 < c < N$ ist schwierig

Der hier aufgeführte Algorithmus aus dem Paper von 1983 ist nicht ausreichend für eine sichere Implementation. Er bietet nur den Grundstein und berücksichtigt noch keinerlei Härtung gegen RSA spezifische Angriffe [Hes12].

2.5.2 Elliptic Curve Cryptography

Unter *Elliptic Curve Cryptography*, kurz *ECC*, fallen alle asymmetrischen Kryptoverfahren welche mithilfe von elliptischen Kurven über endlichen Körpern verschlüsseln. Eine elliptische Kurve ist eine spezielle algebraische Struktur in der Mathematik auf der eine zyklische Gruppe bezüglich der Addition definiert werden kann. Solch eine Gruppe besteht damit aus einem Erzeugerpunkt und den Elementen die durch die definierte Operation mit dem Erzeuger selbst entstehen.

Dieses Verfahren basiert auf der Tatsache, dass die Addition von Elementen mathematisch stets einfach ist. Jedoch ist das Finden eines Koeffizienten $\alpha \in \mathbb{N}$, für einen gegebenen Punkt P auf einer elliptischen Kurve die nicht durch P, sondern durch einen anderen Punkt B erzeugt wird, sodass $\alpha B = P$ gilt, stets schwierig. Alle Einwegfunktionen die auf dem gerade geschilderten Verfahren basieren, können somit auf eine elliptische Kurve übertragen werden. Analog dazu eignen sich natürlich auch Probleme die in zyklischen Gruppen bezüglich der Multiplikation entstehen ebenfalls zur Projektion auf eine elliptische Kurve. Ein häufig verwendetes Problem aus dieser Kategorie ist das Diskreter-Logarithmus-Problem [Men97].

Ein bekanntes hybrides Verschlüsselungsverfahren ist das *Elliptic Curve Integrated Encryption Scheme* kurz *ECIES*. Dieses verwendet unter anderem eine elliptische Kurve zum verschlüsseln des gemeinsamen symmetrischen Schlüssels. Wie in [MED15] beschrieben, läuft das Verfahren folgendermaßen ab:

1. Sender A
    - erstellt ein flüchtiges Schlüsselpaar bestehend aus dem privaten Schlüssel *privA* und dem öffentlichen Schlüssel *pubA* wobei *privA* aus zwei Zufallszahlen x, y besteht mit $x, y \in \mathbb{N}$ und $pubA = privA \cdot G$ mit G als elliptische Kurve.
    - erstellt ein gemeinsames Geheimnis *sec* mithilfe einer geeigneten Key-Agreement-Function wie beispielsweise Deffie-Hellman [Hes12].
    - nimmt *sec* und optionale Parameter als Eingangsvariablen für die Key-Derivation-Function und bekommt als Ergebnis die Konkatenation eines Message Authentication Code (MAC) Keys mit einem symmetrischen Schlüssel *sym*.
    - verschlüsselt die Nachricht *m* mit *sym* und einem geeigneten symmetrischen Verfahren zu der Chiffre *enc*.
    - erzeugt mit der gewählten Message Authentication Code (MAC) Funktion, der chiffrierten Nachricht, dem MAC Key und optionalen Parametern einen *tag*.
    - versendet das Tripel (*pubA, enc, tag*) an den Empfänger.
2. Empfänger B
    - teilt das empfangene Tripel in seine Bestandteile.
    - nimmt den empfangenen Schlüssel *pubA* und seinen eigenen privaten Schlüssel *privB* und multipliziert diese um das gemeinsame Geheimnis zu errechnen.
    - erzeugt mit dem Geheimnis identisch zum Sender den benötigten MAC Key und alle anderen Parameter die auf dem selben Weg vom Sender erzeugt wurden.
    - vergleicht den selbst erzeugten *tag** mit dem empfangenen *tag* und kann so die Integrität des erhaltenen Tripels verifizieren.
    - kann abschließend mit dem durch das Geheimnis erzeugten symmetrischen Schlüssel die Chiffre *enc* entschlüsseln.

2.5.3 ECC vs. RSA

Wie in [AS11] gezeigt wurde sind ECC Verfahren an vielen Stellen besser und effizienter als RSA basierte Verschlüsselung. ECC basierte Algorithmen sind schneller und schwieriger zu brechen bei steigender Schlüssellänge im Vergleich zum RSA. Außerdem kann ein deutlich kürzeres ECC Schlüsselpaar die gleiche Sicherheit eines größeren RSA Schlüsselpaares bieten. Dadurch können ECC Implementationen kleiner und effizienter realisiert werden. Im Bezug auf die Laufzeit konnte in [DM16] gezeigt, dass das RSA Verfahren nur beim Verschlüsseln deutlich schneller arbeitet. Lediglich in den unteren Bereichen des Security Bit Levels kann das RSA Verfahren bei der Entschlüsselung mithalten. Auch

in der Kombination beider Vorgänge übertrumpft das ECC Verfahren das RSA Verfahren mit seiner deutlich konservativeren Wachstumskurve.

ECC Key Size (bits)	RSA Key Size (bits)	Key Size ratio
160 bit	1024 bit	1:6
224 bit	2048 bit	1:9
256 bit	3072 bit	1:12
512 bit	15360 bit	1:30

Abbildung 2.2: Schlüssellängen im Verhältnis [AS11]

Security	Encryption		Decryption		Total	
Bits	ECC	RSA	ECC	RSA	ECC	RSA
80	7.92	0.55	22.88	19.31	30.80	19.87
112	39.70	0.58	26.33	102.03	66.03	102.61
128	58.43	0.56	27.40	209.60	85.84	210.17
144	77.50	0.57	32.15	311.06	109.65	311.63

Abbildung 2.3: Benchmark Zeit in s [DM16]

2.6 Zusammenfassung

In diesem Kapitel wurden Grundlagen vorgestellt und verschiedene Möglichkeiten Funktionalität abzudecken aufgezeigt. Nun folgt eine kurze Begründung, welche Optionen für das hier vorgestellte Konzept am sinnvollsten erscheinen.

Aufgrund der hohen und unnötigen Ressourcenverschwendung, kommt für den Konsensalgorithmus, in der hier verwendeten Blockchain, nur ein Proof-of-Stake Verfahren in Frage. Die vorgestellte Tendermint-Engine eignet sich uneingeschränkt für die Implementierung einer nicht vollständig öffentlichen Blockchain und funktioniert schneller und effizienter.

Die meisten Features von Cloudflare werden in diesem Konzept nicht benötigt. Viel wichtiger ist hier, die Sicherstellung der Unveränderbarkeit von Daten die das IPFS liefert. Wenn eine logische Trennung von Daten und Zugriffsrechten in der hier entwickelten Applikation erfolgen soll, müssen Daten nachweisbar unveränderbar persistiert werden können.

Für die symmetrische Verschlüsselung kann der 3DES-Algorithmus verwendet werden. Da bei der asymmetrischen Verschlüsselung das RSA-Verfahren in fast allen Punkten schlechter abschneidet, sollte für das hier vorgestellte Konzept das ECC-Verfahren verwendet werden.

3 Verwandte Arbeiten

In diesem Kapitel werden einige verwandte Arbeiten und Projekte vorgestellt, welche mithilfe der Blockchain-Technologie die dezentrale Speicherung und den direkten Austausch von Daten realisieren.

3.1 Storj

Storj ist eine Open-Source-Software die eine Applikation bereitstellt, welche Benutzern das Speichern von Dateien in einer dezentralen Cloud ermöglicht. In ihrem Whitepaper bemängeln die Autoren bisherige Konzepte zentralisierter Cloud-Services aufgrund der zugrundeliegenden alten Client-Server Architektur und dem impliziten Vertrauen, welches der Benutzer dem Betreiber entgegenbringen muss. Sie kritisieren, dass Benutzer indem sie diese Services benutzen, darauf hoffen müssen, dass die Betreiber verantwortungsvoll mit ihren Daten umgehen und ihre Verbindungskanäle nicht durch einen Angreifer oder Malware korrumpiert sind, da Dateien in der Regel unverschlüsselt vom Rechner hochgeladen werden [SW14]. Neben dem Vertrauensproblem bietet eine zentrale Architektur immer eine klar definierte Angriffsfläche für eine Vielfalt an Angriffen wie beispielsweise Distributed-Denial-of-Service (DDos) oder Man-in-the-Middle.

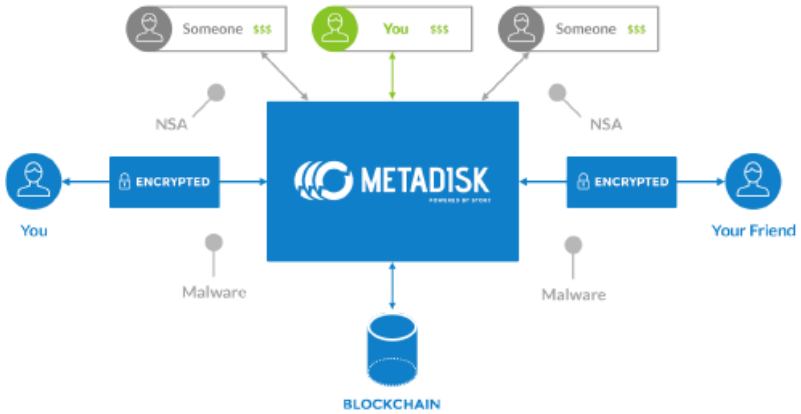

Abbildung 3.1: Storj Netzwerkarchitektur

Das dezentrale Konzept von Storj eliminiert das implizite Vertrauen an einen spezifischen Betreiber, denn jede Datei wird verschlüsselt bevor sie den Rechner auf einem po-

tentiell unsicheren Kanal verlässt. Nur der Besitzer einer Datei kennt den Schlüssel für die Entschlüsselung dieser. Jede Datei wird über ihren einzigartigen Hash identifiziert und kann sowohl vom Netzwerk als auch vom Benutzer selbst auf Veränderungen oder Fehler verifiziert werden ohne den entschlüsselten Inhalt kennen zu müssen [SW14].

Für dieses dezentrale Konzept führt Storj eine eigene netzwerkinterne Kryptowährung ein, mit der Benutzer sich untereinander für Bandbreite und Speicherplatz entschädigen können. Der Preis für entsprechende Leistungen reguliert sich somit fair und neutral über Angebot und Nachfrage. In einem Kostenvoranschlag zeigen die Autoren, dass die Benutzung ihres Netzwerk nur einen Bruchteil des Preises gängiger Cloud-Services kosten würde [SW14].

3.2 Swarm

Das Projekt Swarm ist ein Service auf dem Ethereum Netzwerk welcher Peer-to-Peer Dateiaustausch und ein Content-Delivery-System bereitstellen soll. Swarm kommt ohne eine eigene Kryptowährung aus, da es das Belohnungssystem in verschiedenen Smart Contracts auf die Ethereum Chain ausgelagert. Es ähnelt im Funktionsumfang dem bereits beschriebenen IPFS. Belohnungen für im Netzwerk verrichtete Arbeit gibt es über die Smart Contracts *SWEAR*, *SWAP* und *SWINDLE* [VT16].

Das Akronym *SWAP* steht für *Swarm Accounting Protocol* und bildet ein Konto in einem Smart-Contract ab. Für jeden Teilnehmer im Netzwerk wird hierüber festgehalten wie viel Dateien er angefordert und bereitgestellt hat.[VT16, s. 7] Außerdem bietet *SWAP* eine Funktion an um für erbrachte Leistungen Checks mit entsprechendem Ethereum Gegenwert auszustellen [VT16]. Die Vergütung *off-chain* und nicht als direkte Ether Transaktion umzusetzen hat den Vorteil, dass im Swarm-Netzwerk Bezahlvorgänge unabhängig von der Auslastung und der Geschwindigkeit des Ethereum Netzwerks vollzogen werden können und dieses somit auch gleichzeitig entlastet wird [VT16, p. 9].

Der zweite Smart Contract *SWEAR* steht für *Secure Ways of Ensuring Archival* und ermöglicht Teilnehmern im Netzwerk Langzeitspeicher bereitzustellen. Um eine solche Dienstleistung bereitstellen zu können, muss der Betreiber eines solchen Knotens zuerst eine Pfandzahlung tätigen über welche er für das nicht einhalten von Vertragsbedingungen oder den Verlust von Daten belangt werden kann. Sobald dieser *Collateral* bei *SWEAR* hinterlegt wurde ist der Knoten bevollmächtigt Versprechen im Netzwerk zu geben und für bereitgestellten Speicher Belohnungen einzufahren[VT16, p. 17].

Der letzte Bestandteil des Swarm Ecosystems ist *Secured With Insurance Deposit Litigation and Escrow*, kurz *SWINDLE*. Dieser Smart Contract stellt einen neutrale Instanz dar,

welche über Streitfälle und Entschädigungsansprüche entscheiden soll. Mit diesem kann überprüft werden ob ein Hoster für die Bereitstellung eines bestimmten Dokuments zu belangen ist oder ob dieser fälschlich angeklagt wurde. Dies ist möglich, da alle Transaktionsvorgänge und Verträge in der Blockchain transparent hinterlegt worden sind und somit stets nachvollzogen werden kann, welche spezifischen Verantwortlichkeiten durch die einzelnen Teilnehmer erfüllt werden müssen [VT16, p. 18].

3.3 MeDShare

Sehr ähnlich zu dem in dieser Arbeit vorgestellten Prototypen ist das Projekt MeDShare. Dabei geht es aber nicht um Strafakten oder vertrauliche Anwaltspost, sondern um ebenso sensible Krankenakten und Patientendaten. Die Autoren kritisieren hier vor allem die fehlende Privatsphäre beim Teilen und Verschicken von Krankenakten, da diese problemlos von jedem eingesehen werden können sobald sie in greifbarer Nähe sind [XSA+17]. Weiter könnte kritisiert werden, dass durch das Fehlen einer vollständigen Krankenakte verschiedene Fachärzte nur Bruchteile der Krankheitsgeschichte eines Patienten kennen. Eine durchgängige Akte könnte bei genaueren Diagnosen helfen [XSA+17]. Außerdem ist auch hier jeder Arzt dazu gezwungen für jeden Patienten eine eigene Papierakte anzulegen und diese zu archivieren. Das resultiert in Platz und Papierverschwendung. Die

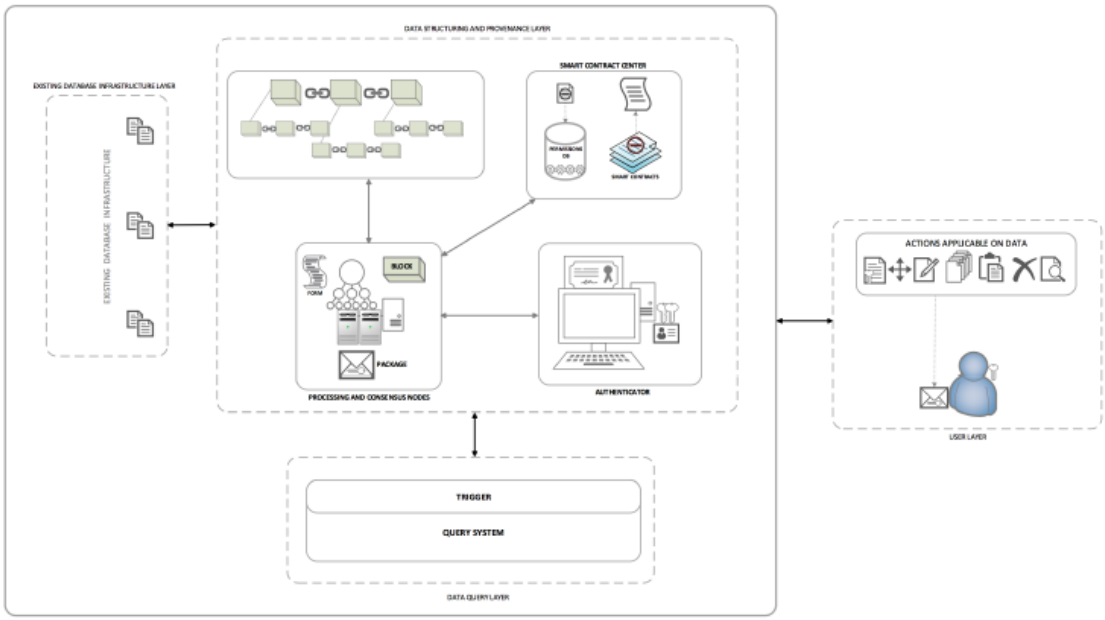

Abbildung 3.2: MeDShare main layer Architektur

Architektur zeigt ein System welches aus vier Hauptbestandteilen aufgebaut ist. Einmal ein *user layer* indem es Berechtigten ermöglicht wird Daten anzufordern und diese zu analysieren. Solche Benutzer können beispielsweise Fachärzte oder Krankenhäuser sein. Weiter sieht man ein *data query layer* mit welchem das Durchsuchen der persistierten

Daten realisiert wird. Außerdem werden hier Trigger implementiert, welche dem Benutzer die direkte Interaktion mit internen Smart Contracts ermöglichen. Weiter ist *ein data structuring and provenance layer* in der Architektur vorgesehen. Dieses hilft mit individuellen Komponenten den Zugang zu angeforderten Daten zu gewährleisten. Außerdem stellt es Algorithmen zur Verfügung um bestimmte Daten explizit zu strukturieren oder zu überwachen. Abschließend existiert ein *database infrastructure layer* mit welchem bereits vorhandene Datenbanken der betreffenden Organisationen angebunden werden können. Da MeDShare nicht für die gesamte Öffentlichkeit vorgesehen ist, sondern sich an eine bestimme Zielgruppe von Berechtigten richtet, wird kein eigene Währung innerhalb des Systems benötigt. Hinzu kommt, dass Speicherplatz vom Netzwerk selbst nicht bereitgestellt wird sondern lediglich die sichere Zugriffsberechtigung auf Daten unternehmensinterner Datenbanken über die Blockchain [XSA+17].

Wenn eine Datenanfrage eines Berechtigten an MeDShare gestellt wird, wird sie vom *data query layer* entgegengenommen und in eine Form gebracht, welche vom *ein data structuring and provenance layer* verstanden werden kann. Nachdem die Legitimität der Anfrage über die Validierung der Signatur überprüft werden konnte, wird die Anfrage zur Bearbeitung weitergeleitet. Ein Smart Contract persistiert als Zwischenschritt den Timestamp der Anfrage gemeinsam mit der Id des Anfragenden, um den Zugriff auf Daten transparent nachvollziehbar machen zu können. Anschließend übernimmt das *database infrastructure layer* die Bearbeitung und stellt die Antwort mithilfe potentiell anzuwendender Regeln für den Benutzer zusammen. Abschließend wird zur zweifelsfreien Nachverfolgbarkeit über Smart-Contracts genau festgehalten welche Daten von wem abgerufen worden sind. Dann werden die angefragten Daten vom System ausgeliefert [XSA+17].

3.4 Smart Will

Bei dem Konzept von Smart Will geht es um einen speziellen Smart Contract, welcher die fälschungssichere Persistierung eines Testaments realisieren soll. Die Autoren erwähnen, dass genau für diesen Anwendungsfall die Kombination aus Blockchain und dem IPFS perfekt sei, da für jedes Dokument vom IPFS ein einmaliger Hash generiert wird und das Dokument zu dem Hash nicht mehr veränderbar ist. Dieser Hash soll dann in einer Blockchain abgespeichert werden. Durch diesen Vorgang wird gewährleistet, dass ein spezifisches Testament für Berechtigte transparent, sicher und unveränderbar persistiert wird [SNK+17]. In Ihrem Paper schlagen die Autoren drei Funktionen vor, die der Smart Contract anbieten soll. Mit *Create Will* soll es möglich sein ein neues Testament aufzusetzen. Dieses wird wie oben beschrieben im IPFS abgelegt. Zusätzlich soll es eine „death“ Flag geben die nur durch den Staat gesetzt werden kann. Sobald diese gesetzt wurde tritt das Testament in Kraft und wird auf der Blockchain veröffentlicht [SNK+17].

Der Ersteller eines Testaments soll die Möglichkeit haben diese jederzeit nach seinen Wünschen anpassen zu können. Dafür wird die Funktion *Update Will* bereitgestellt, welche das bestehende Testament aus dem IPFS lädt und zur Modifikation freigibt. Sobald dies geschieht erlischt die Gültigkeit des Testaments unter dem bisherigen Hash und ein neuer Hash wird nach dem Upload für das Testament generiert, selbst wenn kein Zeichen in dem Dokument verändert wurde [SNK+17].

Als letzte Funktion ist *Probate Will* vorgesehen, welche Personen die in Verbindung mit dem Testament stehen Vermögen zukommen lässt. Dies geschieht über den Public Key des Testamenterstellers und einer validen Todeszertifikatsid [SNK+17]. Möglicherweise ist gerade dieser Ansatz noch etwas praxisfern, da auch physische Gegenstände vererbt werden können, jedoch in Kombination mit einem digitalen Besitzmanagementsystem und der Popularität von Kryptowährungen in naher Zukunft doch möglich.

3.5 Fazit

In diesem Kapitel wurden unterschiedliche Projekte vorgestellt welche mithilfe der Blockchain bestehende Systeme und Arbeitsprozesse optimieren möchten. Auch wenn sie sich durch einzelne Komponenten in ihrer Architektur unterscheiden, so haben sie dennoch eine große Gemeinsamkeit, nämlich die Trennung von Daten und Zugriffsberechtigungen. Die Blockchain ist nicht dafür vorgesehen innerhalb der Blöcke große Datenmengen speichern zu können, da diese an alle Validatoren in einem möglicherweise globalen Netzwerk versendet werden müssen. Eine zu hohe Blockgröße würde die P2P Performanz des Systems zu stark beeinträchtigen. Deswegen werden lediglich benötigte Metadaten, wie beispielsweise Referenzen, Hashwerte oder Schlüssel, die den Zugang zu bestimmten Daten ermöglichen, in der Blockchain festgehalten. Eine weitere Gemeinsamkeit, die auch für das hier vorgestellte Konzept übernommen werden soll, ist die eindeutige Identifikation von Daten über deren Hashwerte.

4 Konzept

In dieser Arbeit soll ein Konzept entwickelt werden, welches zeigt, wie ein vertraulicher Datenaustausch über die Blockchain und mithilfe des IPFS realisiert werden kann. Das Softwarekonzept soll es Benutzern ermöglichen, sich mit dem öffentlichen Schlüssel ihres asymmetrischen Schlüsselpaares in dem Blockchain Netzwerk zu identifizieren und anschließend lokale Daten verschlüsselt an andere Beteiligte versenden zu können. Für das einfachere Verständnis, wird das Konzept anhand eines konkreten Anwendungsszenarios präsentiert.

In diesem Szenario möchten Anwälte Schriftsätze und Akten mit der Staatsanwaltschaft austauschen. Dabei ist von zentraler Bedeutung, dass Integrität, Authentizität und Vertraulichkeit gewährleistet sind. Außerdem müssen Ein- und Ausgangsfristen fälschungssicher registriert werden und zu jedem Zeitpunkt überprüfbar sein. Das Konzept soll außerdem folgende Kriterien erfüllen:

1. Eindeutige Identifikation:
   Jeder Teilnehmer ist über eine einmalige Adresse die ihn pseudonymisiert für alle anderen im Netzwerk sichtbar.

2. Vertrauliche Persistierung:
   Daten, die Benutzer innerhalb des Netzwerks austauschen, werden in verschlüsselter Form abgespeichert.

3. Transparenter und vertraulicher Austausch:
   Der Austauschprozess zwischen Teilnehmern kann immer nachvollzogen werden, die Daten selbst werden jedoch stets verschlüsselt.

4. Redundante Persistierung:
   Ausgetauschte Datensätze werden auf mehreren Hardwareinstanzen abgespeichert um bessere Verfügbarkeit und höhere Ausfallsicherheit gewährleisten zu können.

4.1 Teilnahme am Netzwerk

Da in dem Beispielszenario nur Anwälte und Staatsanwälte teilnehmen können und das Netzwerk somit nicht gänzlich öffentlich betrieben wird, sollte die Zulassung neuer Teilnehmer über eine zentrale Authentifizierungsstelle (Root-CA) erfolgen. Um Benutzer im

Netzwerk eindeutig identifizieren zu können, benötigen diese eine einzigartige öffentliche Adresse und einen privaten Schlüssel. Beides kann über ein 256-Bit ECC Schlüsselpaar abgebildet werden.

Wenn ein neuer Anwalt am Netzwerk teilnehmen möchte, generiert er ein asymmetrisches Schlüsselpaar und übermittelt seinen öffentlichen Schlüssel an die Authentifizierungsstelle, welche diesen für ihn initial signiert. Gemeinsam mit einer gültigen Signatur ist es möglich innerhalb des Netzwerkes ein neues Schlüsselpaar zu registrieren.

Sobald ein signierter Schlüssel in die Blockchain eingetragen wurde, kann er selbst zum signieren neuer Schlüsselpaare verwendet werden. Das bedeutet, dass nicht jeder einzelne Teilnehmer die zentrale Authentifizierungsstelle zur Zulassung verwenden muss. Es reicht somit für eine Kanzlei mit mehreren Anwälten aus, ein einzelnes Schlüsselpaar zu signieren, um damit dann intern beliebig viele Teilnehmer zuzulassen oder auch ohne viel Aufwand temporäre Konten erstellen zu können.

Da der öffentliche Schlüssel unter Umständen sehr lang sein kann, eignet sich ein kürzerer Hashwert besser für die eindeutige Adresse des neuen Teilnehmers. Es wäre möglich, nicht den öffentliche Schlüssel selbst zur Adresse des Anwalts im Netzwerk herzunehmen, sondern den resultierenden Transaktionshash aus dem Registrierungsvorgang der Authentifizierungsstelle. Neben der kürzeren Adresse, stünde damit, der zum Verschlüsseln von Daten benötigte öffentliche Schlüssel des Anwalts, allen Teilnehmern des Netzwerkes über diese Adresse in der Blockchain zur Verfügung.

Der Verlust des zugehörigen privaten Schlüssels resultiert hierbei in einem Komplettverlust der Daten, die über die Anwendung für den öffentlichen Schlüssel geteilt wurden. Dies ist verheerend wenn es ungewollt passiert, jedoch auch der einzige Weg eigene Dateien in der Blockchain indirekt zu löschen. Ein direkter Löschvorgang ist in einer blockchainbasierten Architektur weder erwünscht noch möglich.

4.2 Interaktion mit dem Netzwerk

Teilnehmer benötigen eine grafische Oberfläche, mit der sie intuitiv im Netzwerk interagieren können. Nachdem sich ein Benutzer über sein Schlüsselpaar im Netzwerk identifiziert hat, soll er auf eine grafische Übersicht weitergeleitet werden. Dort stehen ihm Interaktionsmöglichkeiten, wie bereits geteilten Dateien, noch nicht geöffnete Transaktionen und der Versand eigener Dateien an andere Teilnehmer zur Verfügung.

4.3 Integrität und Authentizität geteilter Daten

Eine Möglichkeit der Integritätsverifizierung bei Dateien ist das Prüfsummenverfahren. Hierbei wird mit einem Hash-Algorithmus eine eindeutige Kennung für eine spezifische Datei generiert. Diese könnte, wie in [CMC+18] empfohlen, beispielsweise mit Algorithmen der SHA-2 Familie, über die ganze Transaktion gebildet werden. Für das hier vorgestellte dezentrale Konzept, welches das Vertrauen der Teilnehmer untereinander obsolet machen soll, wäre dieses Verfahren alleine jedoch nicht ausreichend. Angenommen ein Angreifer würde es in irgend einer Weise bewerkstelligen können die Transaktion vor der Übertragung an die Blockchain abzufangen und zu verändern *(Man-in-the-Middle)*, könnte er implizit auch eine neue Prüfsumme bilden. Die Authentizität einer Transaktion ist somit über ein reines Prüfsummenverfahren nicht gegeben. Damit die Authentizität zweifelsfrei gewährleistet werden kann, muss sichergestellt werden, dass die angehängte Prüfsumme vom Ersteller der Transaktion gebildet wurde und keine Manipulation stattgefunden hat. Dieses Problem kann mithilfe einer digitalen Signatur gelöst werden. Das Verfahren benutzt dafür das bereits erstellte asymmetrische Schlüsselpaar. Der private Schlüssel eines Teilnehmers identifiziert diesen Eindeutig in einem Netzwerk. Die gebildete Prüfsumme wird dann vor der Übermittlung zusätzlich mit dem privaten Schlüssel des Erstellers verschlüsselt. Der Empfänger der Transaktion muss dann zusätzlich die verschlüsselte Prüfsumme mit dem öffentlichen Schlüssel des Senders entschlüsseln bevor er diese mit seiner selbst gebildeten Prüfsumme vergleicht. Damit ist nun sowohl die Integrität als auch die Authentizität der Transaktion zweifelsfrei gewährleistet.

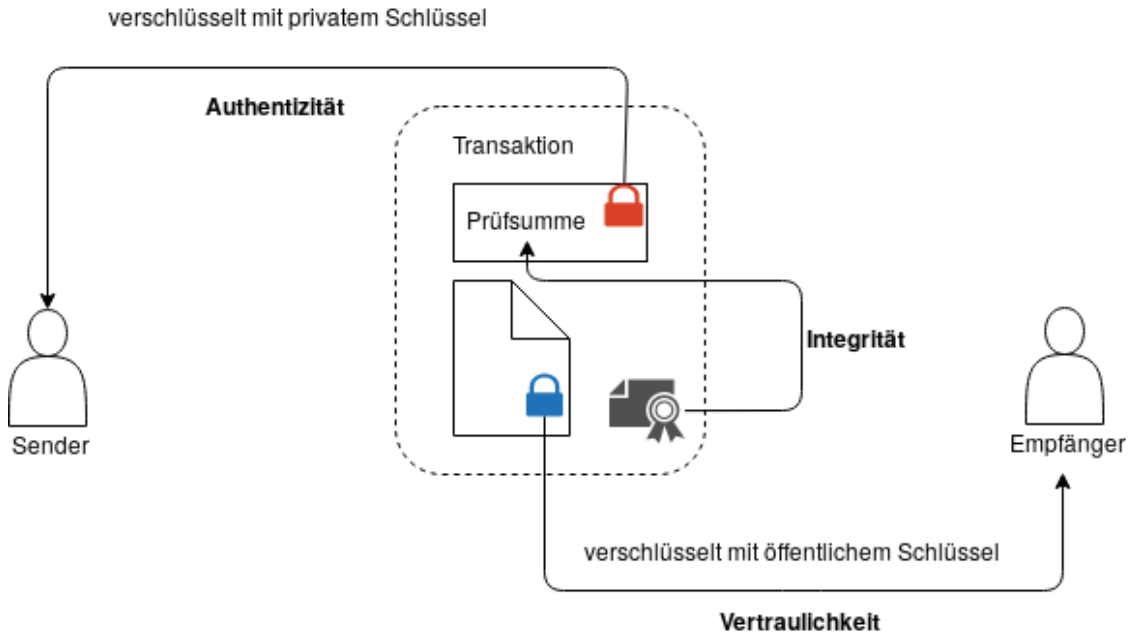

Abbildung 4.1: Vertraulichkeit, Integrität und Authentizität einer Transaktion

4.4 Transaktionsmodell

Wie bereits erwähnt, werden neue Daten stets über Transaktionen in der Blockchain eingetragen. Bisher wurden nur vertrauliche Transaktionen von einem Sender an einen Empfänger beschrieben. In dem Ausgansszenario gibt es jedoch Anwendungsanforderungen, in denen eine Art öffentliche Transaktion vorhanden sein muss, damit Benutzer Änderungen zu ihrem Zustand an alle Teilnehmer im Netzwerk publizieren können. Daraus resultiert die grundlegende Unterscheidung zweier Transaktionsarten, nämlich öffentliche und vertrauliche Transaktionen. Unter der Annahme, dass bis auf die Empfängeradresse, Prüfsumme und den Transaktionstyp, alle Daten vertraulich behandelt werden müssen, kann das Modell für vertrauliche Transaktionen folgendermaßen aussehen:

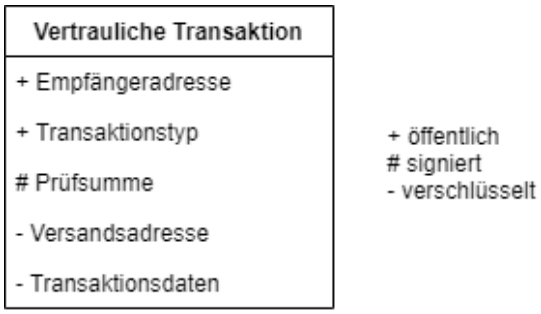

Abbildung 4.2: Vertrauliches Transaktionsmodell

Bei öffentlichen Transaktionen ist nur die Empfängeradresse vorhanden mit der ein Teilnehmer identifiziert werden kann, der Informationen über sich im Netzwerk publizieren möchte. Der Sender einer solchen Transaktion trägt sich selbst als Empfänger ein um Informationen auf sich selbst abzubilden zu können. Die Transaktionsdaten werden nicht verschlüsselt. Durch die Signatur bleiben Integrität und Authentizität der Transaktion erhalten, vertraulich ist sie jedoch nicht mehr.

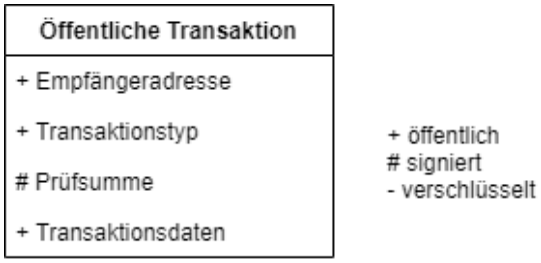

Abbildung 4.3: Öffentliches Transaktionsmodell

4.5 Teilen von Daten

Wenn ein Teilnehmer Daten verschicken möchte, benötigt er die öffentliche Adresse seines Gegenübers. Unter dieser steht ihm der öffentliche Schlüssel seiner Zielperson zur Verfügung, welcher für die Verschlüsselung der Zugriffsberechtigung benötigt wird. Wie in der folgenden Abbildung 4.4 zu sehen ist, benötigt ein Teilnehmer der eine Datei teilen möchte die Daten selbst und die Adresse des Empfängers um den Prozess zu starten. Anschließend wird in der Blockchain überprüft ob der Empfänger bekannt ist. Wenn dies der Fall ist, wird sein öffentlicher Schlüssel ausgelesen, andernfalls wird der Prozess abgebrochen. Im nächsten Schritt wird ein neuer symmetrischer Schlüssel, genau für diesen Zweck und unabhängig von dem bestehenden asymmetrischen Schlüsselpaar, für die zu teilenden Daten generiert.

Wie bereits in dem Kapitel Verwandte Arbeiten erwähnt wurde, ist die Blockchain nicht dafür geeignet große Datenmengen performant innerhalb der Blöcke zu speichern. Deswegen wird die eigentliche Datei symmetrisch verschlüsselt im IPFS abgelegt. Das IPFS liefert einen eindeutigen Hash zur Datei unter dem die Daten abrufbar sind. Abschließend werden Metadaten, der IPFS Link und der dazugehörige symmetrische Schlüssel in einer Transaktion zusammengefasst. Die Transaktion wird bevor sie in der Blockchain publiziert wird asymmetrisch mit dem öffentlichen Schlüssel des Empfängers verschlüsselt. Die Transaktionsdaten können nun über den Hash abgerufen werden.

In der Abbildung 4.5 wird gezeigt wie eine geteilte Datei von einem Berechtigten heruntergeladen werden kann. Hierzu wird lediglich der Transaktionshash benötigt. Mit diesem Hash kann der Empfänger die für ihn bestimmte Transaktion aus der Blockchain laden und diese mit seinem privaten Schlüssel entschlüsseln. In den entschlüsselten Transaktionsdaten ist sowohl der IPFS Link, als auch der dazugehörige symmetrische Schlüssel enthalten. Der Berechtigte kann nun die verschlüsselten Binärdaten aus dem IPFS laden, diese entschlüsseln und mithilfe der mitgelieferten Metadaten wieder in das ursprüngliche Format bringen.

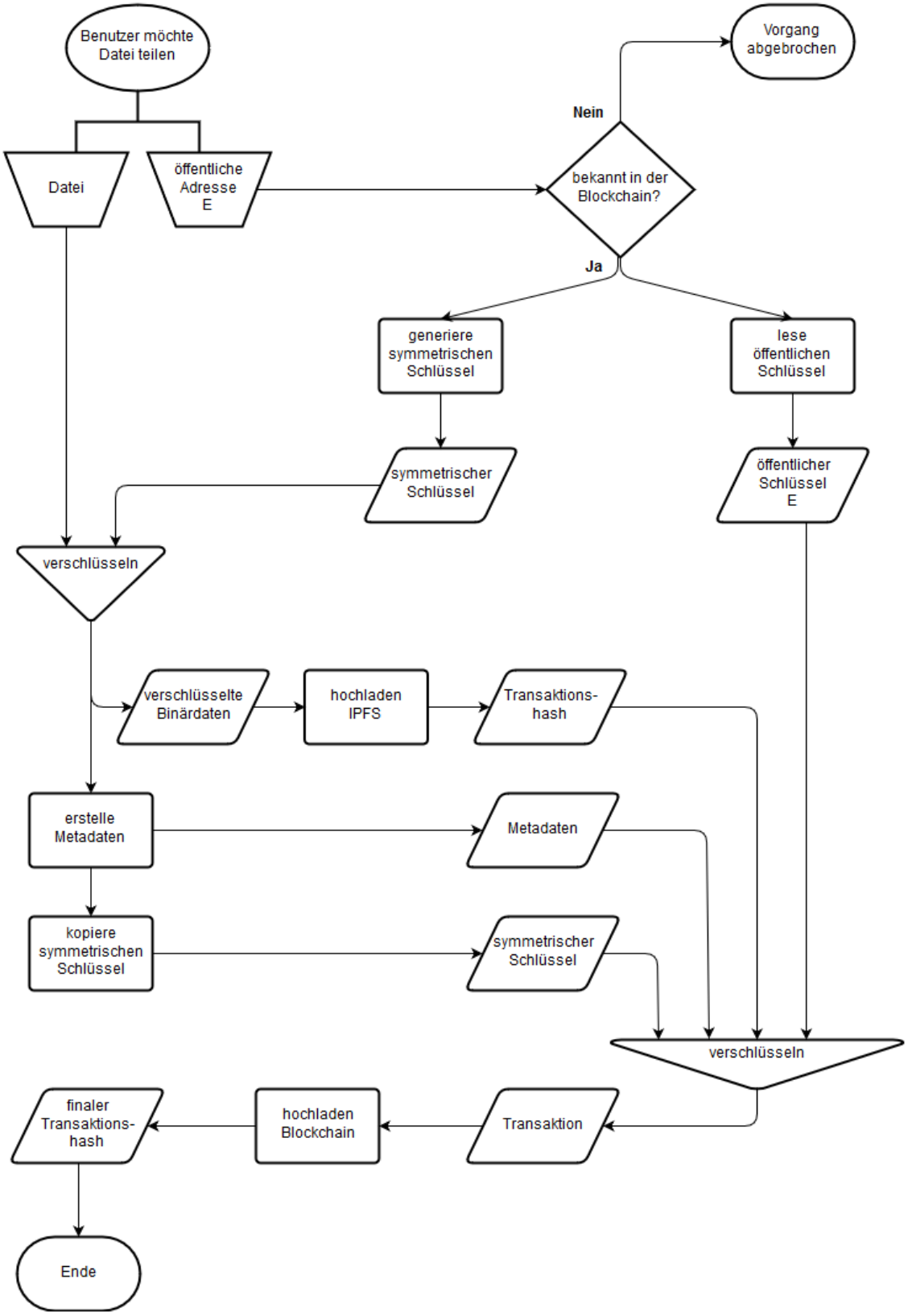

Abbildung 4.4: Teilen einer Datei

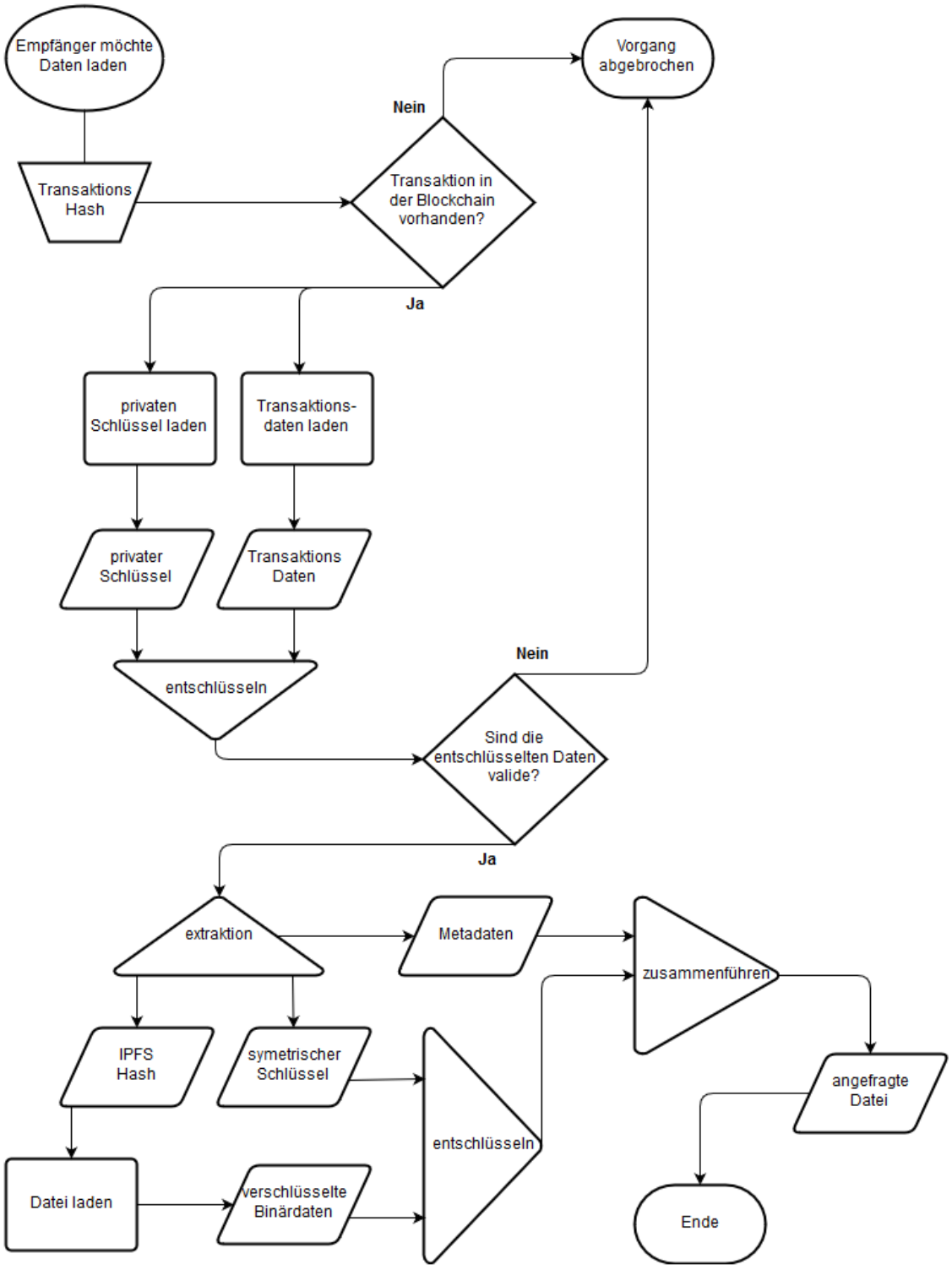

Abbildung 4.5: Empfangen einer Datei

4.6 Ein und Ausgangsfristen

Im juristischen Umfeld hat das Wahren von Firsten unmittelbare rechtliche Konsequenzen. Das Setzen eines Ausgangszeitpunktes ist trivial, da dieser nach der Einstellung der Transaktion automatisch durch den Zustandsautomaten gesetzt werden kann, beziehungsweise implizit anhand der Blockhöhe gesetzt wird. Dadurch wird gleichzeitig sichergestellt, dass der Versandszeitpunkt nicht manuell manipuliert wurde und jederzeit transparent von allen Teilnehmern validiert werden kann. Nicht trivial hingegen ist die Konzeption des Eingangszeitpunktes.

Da nachträgliche Veränderungen an Feldern in der Blockchain nicht möglich sind erfordert das Setzen eines Lesezeitpunktes mehr als eine boolesche Variable. Jede Zusatzinformation zu einer Datei muss in der Blockchain über eine eigene Transaktion abgebildet werden. Rückblickend auf 4.3 kann eine solche Lesebestätigung als öffentliche Transaktion modelliert werden. In dieser würde man beispielsweise den Transaktionshash der angefragten Datei, den Zugriffszeitpunkt und die Adresse des Zugreifenden, vor dem zurückliefern der Antwort, in der Blockchain persistieren. Dieser Prozess ähnelt sehr dem in 3.3 beschriebenem Zugriffs-Kontrollprozess von MeDShare. Dadurch kann zu jedem Zeitpunkt transparent validiert werden, ob eine Eingangsfrist gewahrt wurde.

Wenn der Empfänger der Datei diese das erste mal anfragt, erstellt er automatisch eine solche Empfangstransaktion. Sollte eine solche Transaktion zu einer Datei in der Blockchain nicht gefunden werden können, gab es bisher implizit keinen Zugriff auf die versendeten Daten. Um eine Manipulation dieses Eintrages zu vermeiden, muss von dem implementierten Zustandsautomaten sichergestellt werden, dass solche Empfangstransaktionen nur vom Empfänger der Datei ausgehen können und dass diese, beim Zugriff auf Daten, zwingend gesetzt werden müssen. Das Modell kann folgendermaßen aussehen:

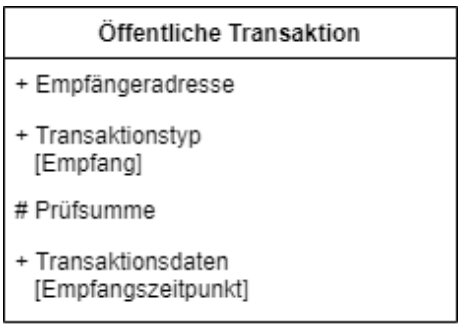

Abbildung 4.6: Empfangstransaktion

4.7 Übertragung von Zugriffsrechten

In dem beschriebenen Anwendungsszenario wäre das Einstellen einer Urlaubsvertretung ein wünschenswertes Feature. Die Anwendung muss also eine zeitlich limitierte Übertragung von Zugriffsrechten ermöglichen. Diese Anforderung könnte über spezielle Adressbuchtransaktionen auf der Blockchain realisiert werden. Diese wären im Bezug auf 4.3 ebenfalls öffentliche Transaktionen. Eine solche Transaktion würde die Adresse des Empfängers, die Adressen der Urlaubsvertretungen und einen Verfallszeitpunkt beinhalten. Die erteilte Zugriffsberechtigung verfällt implizit, wenn der gesetzte Zeitpunkt erreicht ist oder aber eine neuere Adressbuchtransaktion in der Blockchain vorhanden ist. Es gilt grundsätzlich die Transaktion, deren Blockhöhe am höchsten ist. Alle gleichartigen Transaktionen mit niedrigerer Blockhöhe verlieren implizit ihre Gültigkeit, selbst wenn der Verfallszeitpunkt noch nicht eingetreten ist. Das Modell kann folgendermaßen aussehen:

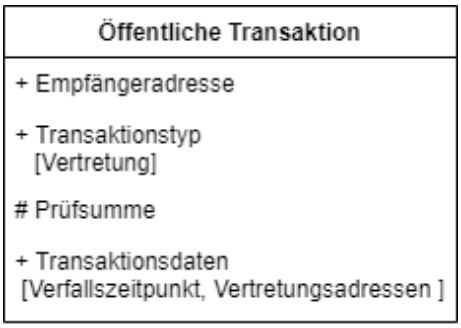

Abbildung 4.7: Adressbuchtransaktion

Angenommen solche Transaktionen sind für einen Teilnehmer vorhanden und eine neue Datei soll versendet werden, dann können folgende Fälle auftreten:

1. Es gibt für den Empfänger keine Adressbuchtransaktion:
   Dies bedeutet für die Anwendung, dass bisher noch keine Urlaubsvertretung vom Empfänger festgelegt wurde. Demnach besteht beim Versenden kein besonderer Handlungsbedarf. Dieser Fall kann ignoriert werden.

2. Es existiert ein Eintrag der auf sich selbst abbildet:
   Dieser Fall tritt ein, wenn vergebene Zugriffsrechte vorzeitig entzogen wurden oder abgelaufen sind. Da ein direkter Löschvorgang nicht möglich ist, muss der Übertragungsvorgang überschrieben werden, indem die Vertretungsadresse wieder auf den Empfänger abbildet. Für die Anwendung besteht auch hier kein besonderer Handlungsbedarf.

3. Es existiert ein Eintrag der auf n andere Adressen abbildet:
   Dieser Zustand repräsentiert das Vorhandensein n aktiver Urlaubsvertretungen. In diesem Fall müssen die zu versendenden Daten allen beteiligten zugänglich gemacht werden. Da in diesem Konzept mehrere Empfänger einer Transaktion nicht vorgesehen sind, muss die Transaktion für den Empfänger und alle eingetragenen Urlaubsvertretungen einzeln erstellt werden. Die Urlaubsvertretungen erhalten dann die Daten so als wären es ihre eigenen. Wenn der eigentliche Empfänger wieder anwesend ist, hat er nachträglich ebenfalls Zugriff auf möglicherweise verpassten Daten.

5 Implementation

In diesem Kapitel wird ein Implementationsvorschlag beschrieben, welcher den Austausch von vertraulichen Dateien im juristischen Umfeld realisieren soll und das vorhergegangene Konzept verwendet. Hierfür wird die Tendermint Blockchain als Verwaltungsinstanz von Zugriffsrechten und Schlüsseln, sowie das IPFS als Content-Delivery-Network der Daten, eingesetzt. Dieses Kapitel gibt einen Überblick über Prozesse und Architekturen, die einen Datenaustausch mithilfe dieser Technologien ermöglichen.

5.1 Systemübersicht

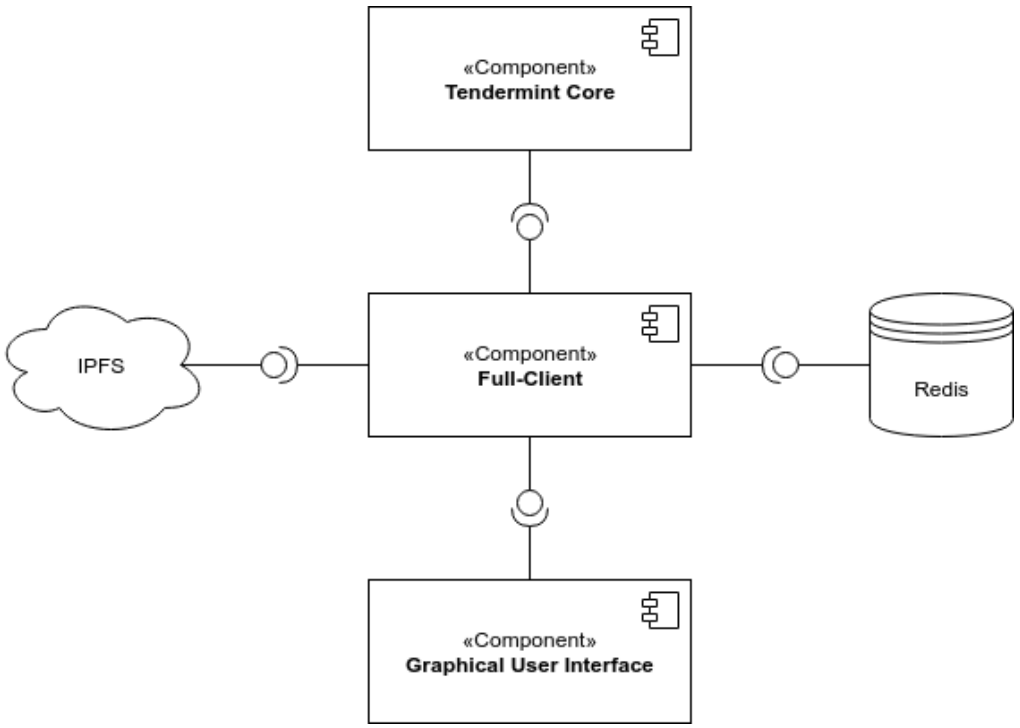

Abbildung 5.1: Konzept Systemübersicht

Wie in 5.1 visualisiert wurde, ist der Full-Client die zentrale Komponente des Systems. Er stellt die Implementierung der ABCI-Schnittstelle für den Tendermint-Core zur Verfügung, verwaltet die Dateninteraktion zwischen der lokalen Persistenzschicht und dem IPFS und liefert gekapselt verfügbare Interaktionsfunktionalität mit allen Komponenten über ein Interface an die grafische Oberfläche.

5.2 Tendermint Core

Eine Tendermint Blockchain besteht aus zwei Hauptkomponenten: Dem *Tendermint Core* und einer *Tendermint Application*. Der *Tendermint Core* stellt Basisfunktionalität wie P2P, Persistenz und das Konsensprotokoll bereit, überlässt die Validierung der Semantik jedoch der individuell implementierten *Tendermint Application*. Damit ein Tendermint Knoten funktioniert müssen beide Bestandteile zwingend in der Lage sein miteinander zu kommunizieren.

Die Kommunikation zwischen den Komponenten verläuft dabei stets unidirektional. Der *Core* stellt Anfragen an die *Application* bezüglich der Semantik und Validierung. Die Trennung der erwarteten Antwortfunktionalität über ein Interface, ermöglicht es Benutzern eine spezifische und programmiersprachenagnostische Implementierung dieser bereitzustellen. Eine Programminstanz die beide Komponenten beinhaltet, gilt als vollwertiger Tendermint-Knoten und kann als Full-Client bezeichnet werden.

Für die externe Interaktion und neue Datensätze, in Form von Transaktionen, wird vom *Tendermint Core* eine Remote Procedure Call (RPC) Schnittstelle bereitgestellt. Anfragen von externen Anwendungen zum allgemeinen Zustand der Blockchain oder gezielten Transaktionen können über diese Schnittstelle gestellt werden. Teilnehmer sollten lokal einen eigenen Tendermint Knoten mit einer ABCI-Applikation betreiben um die beste Performance und Sicherheit gewährleisten zu können. Knoten müssen hierbei nicht zwingend Validatorknoten sein. Theoretisch wäre es möglich, die Benutzeroberfläche strikt getrennt von der Tendermint-Logik zu realisieren und lediglich mit dem bereitgestellten Endpunkt, über HTTP oder ein Websocket, zu kommunizieren.

Dies hätte den Vorteil, dass sich Netzwerkteilnehmer nicht selbst um die Konfiguration und Wartung von Tendermint kümmern müssten. Um am Netzwerk aktiv teilnehmen zu können, würde es reichen die Benutzeroberfläche mit der IP-Adresse und Portnummer eines laufenden Tendermint Knotens zu verbinden. Der Nachteil einer solchen Lösung wäre jedoch der erhöhte Programmieraufwand und Anfrageoverhead um die entkoppelte Instanz immer auf dem aktuellen Stand der Blockchain halten zu können. Außerdem gäbe es dann potentielle Knoten im Netzwerk, welche Schnittstellen zu Tendermint öffentlich und ohne Authentifizierung bereitstellen würden. Da dies ein offensichtliches Sicherheitsrisiko darstellt, wird in diesem Implementationsvorschlag zu einer Realisierung mit einer lokalen Tendermint Instanz geraten.

5.3 Tendermint Full-Client

Um die im Konzept vorgestellte Semantik validieren zu können, muss eine *Tendermint Application* erstellt werden, welche das bereitgestellte *ABCI-Interface* implementiert. Die Architektur eines Full-Clients würde dann folgendermaßen aussehen:

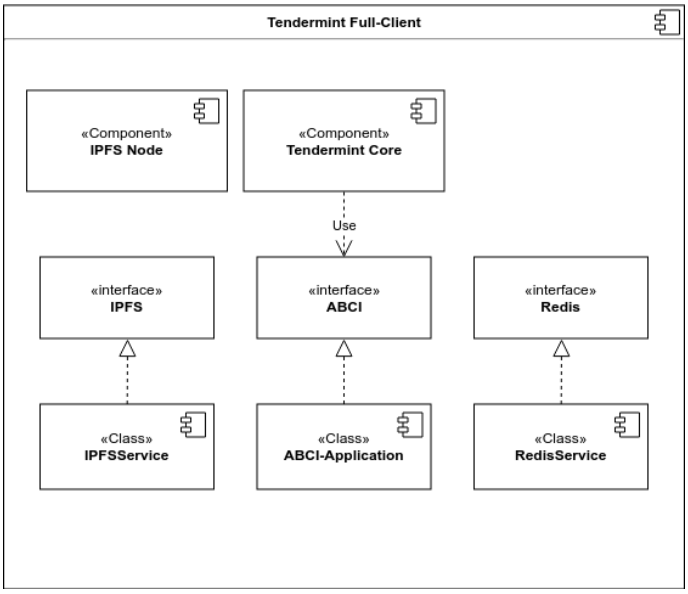

Abbildung 5.2: Tendermint Full-Client

5.3.1 AppState

Um die multiplen Tendermint Full-Clients, die am Netzwerk teilnehmen, stets mit dem aktuellen Zustand der Blockchain synchronisiert halten zu können, benötigt jede teilnehmende Instanz eine Möglichkeit ihren individuellen Zustand eindeutig zu definieren. Diese Funktionalität wird mit dem AppState bereitgestellt, welcher jeweils die vom Client zuletzt gesehene Blockhöhe und den dazugehörigen AppHash persistiert. Bei jedem neuen Block, der Transaktionen enthält, wird dieser auf Basis des vorhergehenden States für alle Teilnehmer im Netzwerkes neu gebildet.

Dadurch kann eindeutig festgestellt werden, ob sich ein bestimmter Knoten auf dem aktuellen Stand des Netzwerkes befindet oder nicht. Sollte ein Client abgeschaltet werden, speichert er seinen letzten AppState ab und sobald er sich wieder im Netzwerk anmeldet, bekommt er automatisch vom Netzwerk die Blocksequenz mitgeteilt, die er in seiner Abwesenheit verpasst hat. Dies gilt im übrigen auch für neue Full-Clients, welche sich üblicherweise mit einem leeren AppHash anmelden. Dieser Vorgang wird bei Tendermint als *Replay* bezeichnet.

5.3.2 Tags

Damit nicht jeder Client über jede einzelne Transaktion iterieren muss, um festzustellen, ob ihn betreffende Adressen in einem der Empfängerfelder stehen, stellt Tendermint sogenannte *Tags* zur Verfügung. Über diese ist es möglich sich, bei der Kommunikation mit dem *Tendermint-Core* über ein Websocket, gezielt auf getaggte Transaktionen zu subscriben. Um dieses Feature nutzen zu können, müssen die Transaktionen von der ABCI-Applikation mit eigens definierten *Tags* versehen werden. Aus den resultierenden Anforderungen des Konzepts, sollten folgende Tags zusätzlich an die Transaktionen geheftet werden, um Änderungen gezielt in jedem Full-Client finden zu können, ohne jeden Block durchsuchen zu müssen:

- Empfängeradresse & Transaktionstyp

5.3.3 ABCI Validierungsprozess

Die Java Implementierung des ABCI Interfaces wurde bereits mit **jABCI** [jTe18] realisiert. Der beschriebene Prozess bezieht sich auf die dort vorhandenen Objekte. Die Erste Methode, welche bei der Anmeldung der Applikation von Tendermint aufgerufen wird, ist stets *Info*. Tendermint erwartet hier die letzte Blockhöhe und den AppHash der lokalen Instanz als Antwort. Die erwartete Antwort kann also einfach in das vom Interface vorgegebene `ResponseInfo` Objekt gepackt werden. Sollten die hier angefragten Informationen noch nicht vorhanden sein, weil die lokale Applikation beispielsweise noch nicht am Netzwerk angemeldet war, kann mit Blockhöhe 0 und einem leeren Bytearray geantwortet werden.

Wenn von Tendermint festgestellt wird, dass der Client sich nicht auf dem aktuellen Stand der Blockchain befindet, liefert der *Tendermint-Core* an dieser Stelle das automatisches Replay aller verpassten Blöcke. Dieses kann jedoch nur erfolgen, wenn mit einem dagewesenen AppHash und der zugehörigen Blockhöhe geantwortet worden ist, da sonst keine Differenz zum bestehenden Zustand gebildet werden kann.

Wenn eine neue Transaktion an den *Tendermint Core* einer Full-Client Instanz gebroadcastet wurde, wird *CheckTx* aufgerufen, mit der die atomare Validierbarkeit der Transaktion überprüft werden soll. Die Methode antwortet mithilfe des `ResponseCheckTx` Objektes, ob die Transaktion gültig ist oder nicht. Sollte dies nicht der Fall sein, wird die betroffene Transaktion verworfen bevor sie an alle anderen Knoten im Netzwerk propagiert wird und im Mempool landet.

Sobald mindestens eine Transaktion im Mempool vorhanden ist, kann sie potentiell in den nächsten Block inkludiert werden. Der Validatorknoten, welcher durch den Tendermint-Algorithmus zum vorschlagen des nächsten Blocks ausgewählt wurde ruft, bevor er

einen neuen Block bildet, die Methode *BeginBlock* auf. Falls es im Anwendungskontext Abhängigkeiten oder Routinen gibt, die vor der Erstellung eines neuen Blocks behandelt werden müssen, kann dies in dieser Methode implementiert werden.

Die Methode *DeliverTx* wird direkt im Anschluss aufgerufen um letzte Anpassungen an den Transaktionen zu ermöglichen, bevor diese endgültig im Block abgespeichert werden. Da an dieser Stelle alle Transaktionen in ihrer Persistierungsreihenfolge vorliegen, sollte eine sequentielle Validierung durchgeführt werden. Zwar wurden die einzelnen Transaktionen atomar in *CheckTx* auf ihre Gültigkeit überprüft, jedoch noch nicht auf Abhängigkeiten die sich möglicherweise aus der Reihenfolge ergeben würden. In dem hier vorgestellten Konzept könnte folgendes Problem auftreten:

Eine Zugriffsberechtigungstransaktion und eine Datentransaktion, die den selben Empfänger betreffen, werden simultan abgeschickt und landen gemeinsam in einem Block. Wenn die Zugriffsberechtigung andere Adressen enthält und in der Sequenz vor der Datentransaktion, die nur an einen Empfänger gesendet wurde, persistiert werden soll, tritt ein ungültiger Zustand ein, denn ab dem Zeitpunkt der gesetzten Zugriffsberechtigung müssen alle folgenden Datentransaktionen an alle Beteiligten versendet werden. Die umgedrehte Reihenfolge beider Transaktionen jedoch würde keinen ungültigen Zustand erzeugen. Sollte ein unzulässiger Zustand durch die Sequenz entstehen, so müssen die betroffenen Transaktionen von *DeliverTx* abgelehnt werden.

Des Weiteren ist *DeliverTx* eine geeignete Stelle um die bereits erwähnten Tags an die einzelnen Transaktionen zu heften und den AppState neu zu berechnen. Da der Tendermint Algorithmus, das regelmäßige Bilden von leeren Blöcken vorsieht, darf die Neuberechnung des AppHashes nur ausgeführt werden, wenn *DeliverTx* aufgerufen wurde und damit implizit Transaktionen im Block vorhanden sind. Ein passender Hash-Algorithmus für den AppHash, wäre an dieser Stelle der bekannte `RIPEMD-160` Algorithmus [DBP96].

Die Methode *EndBlock* wird von Tendermint aufgerufen, wenn ein neuer Block abgeschlossen werden soll. Für das Netzwerk bedeutet das konkret, dass ein weiterer Block hinzukommen soll und sich die Blockhöhe des Netzwerkes um eins erhöht. Tendermint übermittelt in dem `RequestEndBlock` Objekt die neue Blockhöhe an die Applikation, welche diese lokal für den Zustand des Full-Clients aktualisieren sollte.

Nachdem ein neuer Block fertiggestellt worden ist und an die bestehende Blockchain angehängt zu wurde, ruft Tendermint *Commit* auf. Dies ist der letzte Schritt bevor der Algorithmus wieder von vorne anfängt. An dieser Stelle sollte der lokale AppState des Full-Clients aktualisiert werden. Die Entscheidung ob dieser sich verändert hat oder ob

gerade ein leerer Block angehängt wurde, ist implizit über den optionalen Aufruf von *DeliverTx* getroffen worden. Die Höhe jedoch sollte sich zwingend verändert haben, da *EndBlock* auch bei leeren Blöcken aufgerufen wird.

5.3.4 Tendermint RPC Endpoint

Der *Tendermint Core* stellt eine Remote Procedure Call (RPC) Schnittstelle zur Verfügung, über die diverse Interaktionsmöglichkeiten mit Tendermint zur Verfügung stehen. Diese wird sowohl über HTTP als auch über ein Websocket angeboten. Da in dieser Architektur, `subscribe` verwendet werden soll und dieser Endpunkt nicht über HTTP verfügbar ist, muss der Full-Client die RPC-Schnittstelle über ein Websocket benutzen. Weitere Methoden die vom Client verwaltet werden sollten und anschließend der grafischen Benutzeroberfläche über ein Interface zur Verfügung stehen sind:

- `broadcast_tx_sync` um neue Transaktionen zu übertragen
- `tx` um Transaktionen anhand ihres Hashes wiederzufinden
- `dump_consensus_state` um Informationen über den Konsens zu erhalten
- `status` um Informationen zum Status zu erhalten
- `block` um gezielt nach Blöcken suchen zu können

5.3.5 IPFS

Um das IPFS benutzen zu können, ist es zwingend notwendig einen lokalen IPFS-Knoten zu betreiben. Diese Anforderung kann und sollte zusätzlich vom Full-Client übernommen werden, um eine saubere Trennung der Verantwortlichkeiten innerhalb der Gesamtarchitektur zu gewährleisten.

Es muss sichergestellt werden, dass eine Installation einer IPFS Instanz auf der lokalen Maschine vorhanden ist. Sollte dies nicht der Fall sein, kann die Installation vom Full-Client nachgeholt werden. Anschließend sollte der Knoten, vom Full-Client in einem eigenen Thread gestartet werden. Dieser kann nun über eine API für den Austausch von Daten benutzt werden.

Das IPFS ist in diesem Konzept an jeder Transaktion beteiligt, die größere Datenmengen, wie beispielsweise Dateien, enthält. Bevor eine Datentransaktion an die Blockchain publiziert werden kann, müssen die zu teilenden Daten erfolgreich hochgeladen worden sein, um den IPFS-Hashwert innerhalb der Transaktion referenzieren zu können. Der Full-Client ist deshalb neben der Installation, auch für die dauerhafte Aufrechterhaltung der Verbindung zum IPFS verantwortlich.

Daten die in das öffentliche IPFS Netzwerk hochgeladen werden müssen symmetrisch verschlüsselt werden. Ausgehend von einem objektorientierten Ansatz, ist es an dieser Stelle hilfreich eine eigene Datenstruktur zu definieren, welche den symmetrischen Schlüssel des Datensatzes und den IPFS-Link enthält. Diese Klasse kann anschließend asymmetrisch verschlüsselt und über die Blockchain geteilt werden.

5.3.6 Lokale Persitenzschicht

Das Persistieren von applikationsinternen Informationen und Dateien sollte in diesem Konzept möglichst dezentral und entkoppelt von lokalen Speichermedien umgesetzt werden. Es wäre theoretisch möglich alle Dateien und sogar applikationsinterne Daten im IPFS abzulegen und lokal lediglich die Hashwerte aller Referenzen zu persistieren. Dies hätte jedoch zur Konsequenz, dass jeder Datensatz mit jeder Anfrage neu heruntergeladen und entschlüsselt werden müsste und die Anwendung offline, zur Arbeit mit bereits vorhandenen Daten, nicht verwendbar wäre.

Aufgrund dessen schlägt der Autor hier einen Mittelweg vor, bei dem alle bereits angefragten Dateien aus dem IPFS zusätzlich zu deren Referenzen lokal abgespeichert werden. Falls eine Datei bereits lokal vorhanden sein sollte, muss diese nicht ein weiteres Mal heruntergeladen werden. Wichtig ist, dass die lokale Persistenzschicht, in dem hier vorgestellten Konzept, nur aus Performancegründen benötigt wird. Sollte diese beschädigt oder verloren gehen, beziehungsweise die Hardware ausgetauscht werden, reicht das asymmetrische Schlüsselpaar um alle Daten zuerst aus der Blockchain und dann aus dem IPFS wiederherzustellen.

Da die lokale Persistenz keine komplexe Datenbankarchitektur erfordert, sondern als möglichst effizienter Zwischenspeicher fungieren soll, eignet sich eine minimale Datenbank wie Redis [Red15]. Dort können alle Objekte in Listen, sogenannten Buckets oder auch als Key-Value Pair effizient persistiert werden. Da Redis jeden Datensatz im String Format abspeichert, ist die maximale Größe pro Datensatzes auf 512 Megabyte beschränkt. Für die lokale Persistenz sollen dann entsprechende Datenstrukturen vorhanden sein, über die festgelegt werden kann, welche Daten in welcher Form in der Datenbank abgelegt werden. Um die Umwandlung der Klassen in das String-Format und wieder zurück gewährleisten zu können, eignet sich das Jackson-Framework, über welches mithilfe von Annotationen, für einen Redis-Client wie Redisson, in den Klassen kenntlich gemacht werden kann, wie die diese umgewandelt werden sollten.

Aus den bisher beschriebenen Anforderungen sollten folgende Datensätze lokal vorhanden sein und entsprechend modelliert werden:

- Ein Key-Value Pair, welches als Adressbuch fungiert und die Adressen bekannter Kontakte gemeinsam mit ihren öffentlichen Schlüsseln hält. Dieses lokale Adressbuch

würde dann, über das *subscribe* Feature von Tendermint, stets mit dem Zustand der Blockchain synchronisiert werden.

- Das asymmetrische Schlüsselpaar in einem passwortgesicherten Keystore.
- Bereits aus dem IPFS heruntergeladene Daten, als Key-Value Pair, gemeinsam mit dem IPFS-Hash, der Sie global referenziert.

5.3.7 Schnittstelle zur Interaktion über eine GUI

Der Full-Client übernimmt intern die Verwaltung der zentralen Bestandteile dieses Systems und muss seinen Funktionsumfang über eine Schnittstelle zur Benutzung nach außen zur Verfügung stellen. Dazu eignet sich ein eigens definierter Remote-Procedure-Call (RPC) Endpunkt der folgende Methoden zur Interaktion bereitstellt:

- **Name:** *login*
  Beschreibung: Der Full-Client übernimmt die Verwaltung des Schlüsselpaares. Damit der persistierte Keystore entschlüsselt werden kann, muss der Benutzer vor jeglicher Interaktion das Passwort für den Keystore übermitteln.

Parameter	Typ	Benötigt	Beschreibung
password	String	Ja	Passwort für den Keystore

  Antwort: Erfolgreich - OK | Fehlgeschlagen - BAD

- **Name:** *getSharedFiles*
  Beschreibung: Liefert die Metadaten aller empfangenen Dateien mit denen eine Interaktion möglich ist. Die Daten selbst sind noch nicht enthalten.

Parameter	Typ	Benötigt	Beschreibung
identity	String	Ja	Öffentlicher Schlüssel für den Abgleich

  Antwort: Erfolgreich - Liste aller empfangenen Dateien | Fehlgeschlagen - BAD

- **Name:** *getContact*
  Beschreibung: Liefert die Metadaten aller bekannten Kontakte mit denen eine Interaktion möglich ist.

Parameter	Typ	Benötigt	Beschreibung
identity	String	Ja	Öffentlicher Schlüssel für den Abgleich

  Antwort: Erfolgreich - Liste aller Kontakte | Fehlgeschlagen - BAD

- **Name:** *getData*
  Beschreibung: Liefert die konkreten Daten zu einer Datenreferenz. Der Full-Client überprüft hierbei, ob eine Instanz in der lokalen Persistenz gefunden werden kann.

Wenn das nicht der Fall ist, erfolgt eine Abfrage der Daten vom IPFS.

Parameter	Typ	Benötigt	Beschreibung
identity	String	Ja	Öffentlicher Schlüssel für den Abgleich
hash	String	Ja	Datenreferenz

Antwort: Erfolgreich - Angefragte Datei als byte[] | Fehlgeschlagen - BAD

- **Name:** *sendData*
  Beschreibung: Benutzer können hier Daten übermitteln die Sie teilen möchten. Der Full-Client überprüft ob der öffentliche Schlüssel mit dem aktuell geöffneten Keystore übereinstimmt und übernimmt die Verschlüsselung und den Transaktionsprozess.

Parameter	Typ	Benötigt	Beschreibung
data	byte[]	Ja	Daten die geteilt werden sollen
reciever	String	Ja	Öffentliche Adresse des Empfängers
sender	String	Ja	Öffentliche Schlüssel des Senders

Antwort: Erfolgreich - OK | Fehlgeschlagen - BAD

- **Name:** *vacation*
  Beschreibung: Benutzer können hier eine Urlaubsvertretung einstellen. Die Urlaubsvertretung muss in der Liste der bekannten Kontakte enthalten sein.

Parameter	Typ	Benötigt	Beschreibung
sender	String	Ja	Öffentlicher Schlüssel für den Abgleich
reciever	String	Ja	Datenreferenz
end	long	Nein	Verfallszeitpunkt der Bertechtigung

Antwort: Erfolgreich - OK | Fehlgeschlagen - BAD

- **Name:** *register*
  Beschreibung: Benutzer die noch kein Schlüsselpaar besitzen können hier ihren signierten Keystore an den Full-Client publizieren.

Parameter	Typ	Benötigt	Beschreibung
keystore	byte[]	Ja	Asymmetrisches Schlüsselpaar
signature	byte[]	Ja	Signatur des Keystores der ROOT CA

Antwort: Erfolgreich - OK | Fehlgeschlagen - BAD

6 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde ein Konzept für den Austausch sensibler Daten über dezentrale Cloud-Speichersysteme entwickelt. Die Verwaltung der Zugriffsrechte dieser wurde mithilfe einer Blockchain modelliert. Als konkretes Anwendugsszenario wurde hierbei der Austausch von Schriftsätzen unter Anwälten angenommen. Im Folgenden Kapitel werden die zentralen Inhalte nochmals zusammengefasst und abschließend ein Ausblick auf mögliche Ansatzpunkte zur Fortführung und Verbesserung dieser Arbeit gegeben.

6.1 Zusammenfassung

Zunächst wurde auf die grundlegende Problemstellung aufmerksam gemacht. Daraus wurden Anforderungen an eine mögliche Lösung erarbeitet. Das Konzept sollte mindestens folgende Kriterien erfüllen:

1. Eindeutige Identifikation von Teilnehmern
2. Vertrauliche Persistierung von Daten
3. Transparenter und vertraulicher Austausch von Daten
4. Redundante Persistierung von Daten

Um fundierte Aussagen zu spezifischen Technologien treffen zu können wurden verschiedene Technologien vorgestellt und deren Vor- und Nachteile im Bezug auf die hier entstehenden Anforderungen eruiert. Des Weiteren wurden Verwandte Arbeiten herangezogen um Lösungsmöglichkeiten ähnlicher Problemstellungen näher zu beleuchten.

Zentrale Technologien, die in dieser Arbeit für eine Lösung miteinander verknüpft worden sind, waren Blockchain, Cloud und Kryptographie. Es wurde ein Prozess entwickelt der Beteiligten mithilfe dieser Komponenten das direkte Teilen und Laden von Daten ermöglicht. Hierfür wurde gezeigt wie neue Teilnehmer in das Netzwerk integriert werden können und die Integrität und Authentizität geteilter Daten sichergestellt werden kann. Da in einer blockchainbasierten Datenbasis Änderungen nur über neue Einträge vorgenommen werden können, wurde ein generisches Transaktionsmodell entwickelt, welches in der Lage ist die verschiedenen Anforderungen innerhalb des Netzwerkes abzudecken.

Zusätzlich ging der Autor auf zentrale Anforderungen eines konkreten Anwendungsszenarios ein und hat gezeigt wie die Wahrung von Ein- und Ausgangsfristen gewährleistet werden könnte und wie im Falle einer Urlaubsvertretung die Übertragung von Zugriffsrechten modelliert werden kann.

Zu dem Konzept wurde in dieser Arbeit ein konkreter Implementierungsvorschlag erarbeitet. Dieser verknüpft Tendermint als Blockchain-Framework mit dem dezentralen Cloud-Speichersystem IPFS. Zur Verschlüsselung wird ein gemischtes Verfahren basierend auf elliptischen Kurven und dem symmetrischen 3DES-Algorithmus eingesetzt. Die aufgestellten Kriterien wurden in dem Implementierungsvorschlag folgendermaßen erfüllt:

1. Teilnehmer konnten über ein asymmetrischen Schlüsselpaar eindeutig identifiziert werden.

2. Daten wurden vertraulich persistiert, da sie vor dem Versandt symmetrisch verschlüsselt worden sind.

3. Der Austausch konnte transparent über die Blockchain und die modellierten Ein- und Ausgangsfristen validiert werden. Daten konnten vertraulich geteilt werden, da symmetrische Schlüssel für eine Transaktion asymmetrisch verschlüsselt worden sind.

4. Daten wurden implizit redundant durch die Verwendung des IPFS-Protokolls persistiert.

6.2 Ausblick

Die hier vorgestellte Arbeit bietet Möglichkeiten zur Verbesserung und Erweiterung, die nicht unerwähnt bleiben sollen. Da der Umgang mit sensiblen Daten beschrieben wird, müssen bei einer wirklichen Implementierung des Konzepts vor allem die Bereiche Datenschutz und langfristige Datensicherheit im Vordergrund stehen.

6.2.1 Post-Quanten-Kryptographie

Noch nicht in Reichweite, aber eine ständige Bedrohung, vor allem für gängige asymmetrischen Verfahren, ist die Weiterentwicklung des Quantencomputers. Das Bundesamt für Sicherheit erwartet, dass mit einer Million Qbits ein 2048 Bit RSA Schlüssel innerhalb von 100 Tagen berechnet werden kann [BSI18]. Da alle Daten in dem hier beschriebenen Konzept in das IPFS geladen werden und eine nachträgliche Anpassung nicht mehr möglich ist, könnte ausgehend von der Benutzung des öffentlichen IPFS Netzwerkes hier der Datenschutz nicht mehr gewährleistet werden. Möglicherweise ist eine Implementie-

rung dieses Konzepts mit quantensicheren Algorithmen zu empfehlen und ein privates IPFS-Netzwerk satt der öffentlichen Instanz zu benutzen.

6.2.2 Hardware Schlüsselpaare

In dieser Arbeit wurden asymmetrische Schlüsselpaare einfach generiert und auf dem Rechner abgespeichert. Das Schlüsselpaar eines Teilnehmers spielt eine zentrale Rolle, da der Besitzer damit Zugang zu allen Daten bekommt und umgekehrt beim Verlust Zugang zu allen Daten verliert. Ein Schlüsselpaar das lokal in einem passwortgeschützten Keystore abgelegt wurde, ist beispielsweise nicht sicher wenn der betroffene PC mit einem Keylogger infiziert ist. Eine mögliche Verbesserung wäre die Unterstützung eines Hardwareschlüssels anzubieten. Der große Vorteil dieser Variante ist, dass der private Schlüssel niemals von der Hardwareinstanz nach außen versendet wird und es keine Möglichkeit gibt diesen auszulesen. So wäre die Benutzung der hier vorgestellten Software auch auf einem unsicheren Gerät stets gewährleistet.

6.2.3 Integration in bestehende Software

Die hier beschriebene Architektur weist eine klare Trennung von Logik und grafischer Benutzeroberfläche auf. Es würde somit die Möglichkeit bestehen, die angebotene RPC Schnittstelle innerhalb vorhandener Datenverwaltungssoftware zu benutzen und Teilnehmern so eine nahtlose Integration dieser Komponente in eine ihnen bereits vertraute Umgebung zu ermöglichen.

Literaturverzeichnis

[AS11] AMARA, M. ; SIAD, A.: Elliptic Curve Cryptography and its applications. In: *International Workshop on Systems, Signal Processing and their Applications, WOSSPA*, 2011, S. 247–250

[Ben18] BENET, Juan: *IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3)*. https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf. Version: 2018, Abruf: 2018-04-15

[BSI18] BSI: *Entwicklungsstand Quantencomputer*. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Quantencomputer/P283_QC_Studie.pdf. Version: 2018, Abruf: 2018-12-27

[Bun18] BUNDESRECHTSANWALTSKAMMER: *Das besondere elektronische Anwaltspostfach*. https://bea.brak.de/. Version: 2018, Abruf: 2018-10-10

[But14] BUTERIN, Vitalik: *A NEXT GENERATION SMART CONTRACT DECENTRALIZED APPLICATION PLATFORM*. https://cryptorating.eu/whitepapers/Ethereum/Ethereum_white_paper.pdf. Version: 2014, Abruf: 2018-10-22

[Car17] CARLOZO, Lou: What is blockchain? In: *Journal of Accountancy* 224 (2017), 07, Nr. 1, 29. https://search-1proquest-1com-100c19cgm01d4.emedien3.sub.uni-hamburg.de/docview/1917635714?accountid=11262. ISBN 00218448

[CMC+18] CHERUBINI, Mauro ; MEYLAN, Alexandre ; CHAPUIS, Bertil ; HUMBERT, Mathias ; BILOGREVIC, Igor ; HUGUENIN, Kévin: Towards Usable Checksums: Automating the Integrity Verification of Web Downloads for the Masses. (2018)

[CV17] CACHIN, Christian ; VUKOLIC, Marko: Blockchain Consensus Protocols in the Wild. In: *CoRR* abs/1707.01873 (2017). http://arxiv.org/abs/1707.01873

[DBP96] DOBBERTIN, Hans ; BOSSELAERS, Antoon ; PRENEEL, Bart: RIPEMD-160: A strengthened version of RIPEMD. In: GOLLMANN, Dieter (Hrsg.): *Fast Softwa-*

re Encryption, Springer Berlin Heidelberg, 1996. – ISBN 978–3–540–49652–6, S. 71–82

[DM16] DINDAYAL MAHTO, Dilip Kumar Y. Danish Ali Khan K. Danish Ali Khan: Security Analysis of Elliptic Curve Cryptography and RSA. In: *Proceedings of the World Congress on Engineering* 1 (2016), June. http://www.iaeng.org/publication/WCE2016/WCE2016_pp419-422.pdf. – ISSN 2078–0966

[Gol18a] GOLEM: *Das Anwaltspostfach kommt mit Sicherheitslücken*. https://www.golem.de/news/bea-das-anwaltspostfach-kommt-mit-sicherheitsluecken-1809-136346.html. Version: 2018, Abruf: 2018-10-10

[Gol18b] GOLEM: *Bundesrechtsanwaltskammer verteilt HTTPS-Hintertüre*. https://www.golem.de/news/bea-bundesrechtsanwaltskammer-verteilt-https-hintertuere-1712-131845.html. Version: 2018, Abruf: 2018-10-10

[Hes12] HESS, Andreas Stein Sandra L. Florian Stein S. Florian Stein: The Magic of Elliptic Curves and Public-Key Cryptography. In: *Jahresbericht der Deutschen Mathematiker-Vereinigung* 114 (2012), Jun, Nr. 2, S. 59–88

[IB17] IDDO BENTOV, Alex M. Ariel Gabizon G. Ariel Gabizon: *Cryptocurrencies without Proof of Work*. https://arxiv.org/abs/1406.5694v9. Version: 2017, Abruf: 2018-10-23

[Inc17] INC, Cloudflare: *A global content delivery network with unique performance optimization capabilities*. https://www.cloudflare.com/static/media/pdf/cloudflare-whitepaper-cdn.pdf. Version: 2017, Abruf: 2018-10-27

[jTe18] JTENDERMINT: *Java implementation of the Tendermint ABCI*. https://github.com/jTendermint/jabci. Version: 2018, Abruf: 2018-10-15

[Jus16] JUSTIZ, Bundesministerium der: *Geschäftsbelastungen bei Gerichten und Staatsanwaltschaften*. https://www.bundesjustizamt.de/DE/SharedDocs/Publikationen/Justizstatistik/Geschaeftsentwicklung_Gerichte_Staatsanwaltschaften.pdf?__blob=publicationFile&v=13. Version: 1999-2016, Abruf: 2018-03-14

[Law17] LAWGEEX: *Comparing the Performance of Artificial Intelligence to Human Lawyers in the Review of Standart Business Contracts*. https://www.lawgeex.com/AIvsLawyer. Version: 2017, Abruf: 2018-03-14

[MED15] MARTÍNEZ, V. G. ; ENCINAS, L. H. ; DIOS, A. Q.: Security and Practical Considerations When Implementing the Elliptic Curve Integrated Encryption Scheme. In: *Cryptologia* 39 (2015), Nr. 3, 244-269. http://dx.doi.org/10.1080/01611194.2014.988363. – DOI 10.1080/01611194.2014.988363

[Men97] MENEZES, Alfred J. ; VAN OORSCHOT, Paul C. (Hrsg.): *Handbook of applied cryptography*. Boca Raton u.a. : CRC Press, 1997 (CRC Press series on discrete mathematics and its applications). – ISBN 0849385237 9780849385230. – Literaturverz. S. 703 - 754

[MSD+06] MORENO, B. M. ; SALVADOR, C.E. P. ; DOMINGO, M. E. ; PEÑA, I. A. ; EXTREMERA, V. R.: On content delivery network implementation. In: *Computer Communications* 29 (2006), Nr. 12, 2396 - 2412. http://dx.doi.org/https://doi.org/10.1016/j.comcom.2006.02.016. – DOI https://doi.org/10.1016/j.comcom.2006.02.016. – ISSN 0140–3664

[Mul17] MULTIADDR: *Multiformat Protocol*. https://multiformats.io/multiaddr/. Version: 2017, Abruf: 2018-10-27

[Nak08] NAKAMOTO, Satoshi: *Bitcoin: A Peer-to-Peer Electronic Cash System*. https://bitcoin.org/bitcoin.pdf. Version: 2008, Abruf: 2018-10-22

[Red15] REDISLABS: *Redis*. https://redis.io/. Version: 2015, Abruf: 2018-11-04

[RSA83] RIVEST, R. L. ; SHAMIR, A. ; ADLEMAN, L.: A Method for Obtaining Digital Signatures and Public-key Cryptosystems. In: *Commun. ACM* 26 (1983), Januar, Nr. 1, 96–99. http://dx.doi.org/10.1145/357980.358017. – DOI 10.1145/357980.358017. – ISSN 0001–0782

[Sch90] SCHNEIDER, Fred B.: Implementing Fault-tolerant Services Using the State Machine Approach: A Tutorial. In: *ACM Comput. Surv.* 22 (1990), Dezember, Nr. 4, 299–319. http://dx.doi.org/10.1145/98163.98167. – DOI 10.1145/98163.98167. – ISSN 0360–0300

[SNK+17] SREEHARI, P. ; NANDAKISHORE, M. ; KRISHNA, G. ; JACOB, J. ; SHIBU, V. S.: Smart will converting the legal testament into a smart contract. In: *2017 International Conference on Networks Advances in Computational Technologies (NetACT)*, 2017, S. 203–207

[Spi11] SPITZ, Stephan: *Kryptographie und IT-Sicherheit : Grundlagen und Anwendungen*. 2. Wiesbaden : Vieweg + Teubner, 2011

[SW14] SHAWN WILKINSON, Jim L.: *Metadisk: Blockchain-Based Decentralized File Storage Application*. https://storj.io/metadisk.pdf. Version: 2014, Abruf: 2018-04-22

[Ten18] TENDERMINT: *Tendermint Documentation*. https://media.readthedocs.org/pdf/tendermint/master/tendermint.pdf. Version: 2018, Abruf: 2018-04-15

[VT16] VIKTOR TRÓN, Dániel a. Nagy Zsolt Felföldi Nick J. Aron Fischer F. Aron Fischer: *swap, swear and swindle incentive system for swarm*. http://swarm-gateways.net/bzz:/theswarm.eth/ethersphere/orange-papers/1/sw^3.pdf. Version: 2016, Abruf: 2018-04-22

[XSA+17] XIA, Q. ; SIFAH, E. B. ; ASAMOAH, K. O. ; GAO, J. ; DU, X. ; GUIZANI, M.: MeDShare: Trust-Less Medical Data Sharing Among Cloud Service Providers via Blockchain. In: *IEEE Access* 5 (2017), S. 14757–14767. http://dx.doi.org/10.1109/ACCESS.2017.2730843. – DOI 10.1109/ACCESS.2017.2730843

Abbildungsverzeichnis

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.
Ich bin mit einer Einstellung in den Bestand der Bibliothek des Fachbereiches einverstanden.

Hamburg, den _____ Unterschrift: _____