Bachelorarbeit

im Studiengang
Wirtschaftsinformatik und digitale Medien

Ein sicheres Peer-to-Peer-Instant-Messaging-Protokoll unter Einbeziehung von Blockchain-Technologie

vorgelegt von

Nicole Sauer

an der Hochschule der Medien Stuttgart

am 29.01.2024

zur Erlangung des akademischen Grades eines

Bachelor of Science

Erstprüfer	Prof. Dr. Peter Thies
Zweitprüfer	Prof. Dr. Stephan Wilczek

Ehrenwörtliche Erklärung

Hiermit versichere ich, Nicole Sauer, ehrenwörtlich, dass ich die vorliegende Bachelorarbeit mit dem Titel: „Ein sicheres Peer-to-Peer-Instant-Messaging-Protokoll unter Einbeziehung von Blockchain-Technologie“ selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Ebenso sind alle Stellen, die mit Hilfe eines KI-basierten Schreibwerkzeugs erstellt oder überarbeitet wurden, kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ich habe die Bedeutung der ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen (§ 24 Abs. 2 Bachelor-SPO), der HdM einer unrichtigen oder unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen.

Leutenbach, den 29.01.2024 N. Sauer

Ort, Datum | Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Kurzfassung

Diese Bachelorarbeit präsentiert die Entwicklungsarbeit an einem innovativen Peer-to-Peer-Instant-Messaging-Protokoll, das auf einer Kombination von Kademlia und Interactive Connectivity Establishment (ICE) basiert. Der Fokus liegt auf der Schaffung eines dezentralen, robusten Peer-to-Peer Kommunikationsprotokolls, das unabhängig von zentralen Servern agiert.

Die Authentifizierung der Teilnehmer erfolgt über öffentliche Schlüssel, die sicher in der Ethereum Blockchain hinterlegt sind. Diese Blockchain-basierte Authentifizierung gewährleistet eine vertrauenswürdige Identitätsüberprüfung, wobei die dezentrale Natur von Ethereum eine hohe Sicherheit und Manipulationssicherheit bietet.

Ein zentraler Beitrag dieser Arbeit ist die Integration von Kademlia, einem verteilten Peer-to-Peer Routing-Algorithmus, und ICE, einem Mechanismus zur Überquerung von Netzwerkgrenzen. Diese Kombination ermöglicht eine effiziente und zuverlässige P2P-Kommunikation, wodurch die Abhängigkeit von zentralen Servern minimiert wird. Die Implementierung wurde darauf ausgerichtet, eine hohe Skalierbarkeit und Robustheit des Systems sicherzustellen.

Die präsentierte Lösung bietet nicht nur eine sichere Peer-to-Peer Kommunikationsumgebung, sondern betont auch die Bedeutung der Blockchain-Technologie für die Authentifizierung in dezentralen Systemen. Das entwickelte Protokoll demonstriert, wie eine Kombination von Ethereum, Kademlia und ICE die Grundlage für eine vertrauenswürdige und effiziente Peer-to-Peer-Instant-Messaging-Infrastruktur schaffen kann.

Abstract

This bachelor's thesis introduces the development of an innovative Peer-to-Peer Instant Messaging protocol, leveraging a combination of Kademlia and Interactive Connectivity Establishment (ICE). The primary focus lies in creating a decentralized, robust Peer-to-Peer communication protocol that operates independently of central servers.

Participant authentication is achieved through public keys securely stored in the Ethereum blockchain. This blockchain-based authentication ensures a trustworthy identity verification process, with Ethereum's decentralized nature providing high security and tamper resistance.

A significant contribution of this work is the integration of Kademlia, a distributed Peer-to-Peer routing algorithm, and ICE, a mechanism for traversing network boundaries. This combination enables efficient and reliable P2P communication, minimizing reliance on central servers. The implementation is designed for high scalability and system robustness.

The presented solution not only offers a secure Peer-to-Peer communication environment but also underscores the importance of blockchain technology for authentication in decentralized systems. The developed protocol demonstrates how a combination of Ethereum, Kademlia, and ICE can form the foundation for a trusted and efficient Peer-to-Peer Instant Messaging infrastructure.

Kapitel 1

Hintergrund und Motivation

In den letzten Jahren hat sich die digitale Kommunikation signifikant gewandelt. 2013 war das Short Message System (SMS) noch dominierend, doch in den darauf folgenden Jahren verlagerte sich der Fokus deutlich hin zum Mobile-Instant-Messaging. Die Nutzung von Instant-Messaging unter der deutschen Bevölkerung stieg von etwa 24% im Jahr 2013 auf beeindruckende 73% im Jahr 2017 (Nier 2017). Die Gründe für diesen Wandel sind vielfältig. Instant-Messaging ist in der Regel kostenlos, schnell und einfach zu bedienen. Darüber hinaus bietet es viele nützliche Funktionen wie Gruppenchats, Sprach- und Videoanrufe, Dateiübertragung und vieles mehr. Instant-Messaging ist zu einem wichtigen Bestandteil unseres täglichen Lebens geworden. Es ist ein wesentlicher Bestandteil der Kommunikation zwischen Freunden und Familie, aber auch zwischen Kollegen und Geschäftspartnern.

Aber Instant-Messaging ist nicht gleich Instant-Messaging. Es gibt verschiedene Arten von Instant-Messaging-Diensten, die sich in ihrer Funktionsweise und ihren Eigenschaften unterscheiden. Die bekanntesten sind zentralisierte und dezentrale Instant-Messaging-Dienste. Zentralisierte Dienste werden von einem zentralen Server verwaltet, der die Kommunikation zwischen den Benutzern vermittelt. Dezentrale Dienste hingegen nutzen eine Peer-to-Peer-Infrastruktur, bei der die Kommunikation direkt zwischen den Benutzern stattfindet.

Zentralisierte Dienste sind in der Regel einfach zu bedienen und bieten eine gute Benutzererfahrung. Sie sind jedoch anfällig für Sicherheitsbedrohungen, da die Kommunikation über einen zentralen Server vermittelt wird. Sollte dieser Server kompromittiert werden, könnte die Kommunikation der Benutzer abgefangen oder manipuliert werden. Darüber hinaus könnten die Betreiber des Dienstes die Kommunikation ihrer Benutzer überwachen und speichern. Dies stellt ein großes Problem für die Privatsphäre der Be-

nutzer dar.

Dezentrale Dienste hingegen haben das Potenzial mehr Sicherheit und Privatsphäre für ihre Nutzer zu bieten, da die Kommunikation direkt zwischen den Benutzern stattfindet. Es gibt keinen zentralen Server, der angegriffen werden könnte. Darüber hinaus können die Betreiber des Dienstes die Kommunikation nicht überwachen, da sie nicht an der Kommunikation beteiligt sind. Dezentrale Dienste sind jedoch in der Regel komplexer und schwieriger zu bedienen als zentralisierte Dienste. Und auch sie sind nicht perfekt. Auch in dezentralen Diensten können Sicherheitslücken auftreten, die die Sicherheit und Privatsphäre der Benutzer gefährden.

Durch die im Jahr 2013 von Edward Snowden veröffentlichten Dokumente wurde deutlich, dass die Kommunikation von Millionen von Menschen von Geheimdiensten überwacht wurde (Greenwald 2013). Dadurch rückten die Themen Sicherheit und Privatsphäre in den Fokus der Öffentlichkeit. Durch diese Enthüllungen entstand zum Beispiel das Peer-to-Peer-Instant-Messaging-Protokoll *Tox*. Es wurde von einer Gruppe von Entwicklern ins Leben gerufen, die sich zum Ziel gesetzt haben, ein sicheres und leicht zu bedienendes Instant-Messaging-Protokoll zu entwickeln, das die Privatsphäre seiner Benutzer schützt (*Tox - The Tox Project* o. D.). Die Entwicklung eines solchen Instant-Messaging-Protokolls ist jedoch nicht leicht. Es müssen viele Aspekte beachtet werden, die in zentralisierten Protokollen nicht vorhanden sind und die die Komplexität des Protokolls erhöhen. Im Verlauf dieser Bachelorarbeit werden verschiedene Aspekte der Peer-to-Peer-Kommunikation, einschließlich Sicherheit, Effizienz und Benutzerfreundlichkeit, beleuchtet. Dabei werden auch bereits verfügbare Protokolle betrachtet, um eine Grundlage für die Entwicklung eines eigenen Protokolls zu schaffen. Darüber hinaus hat die Blockchain das Potenzial, die Integrität und Authentizität von Daten und Dokumenten in Instant Messaging-Plattformen sicherzustellen, weshalb auch diese Technologie in dieser Arbeit betrachtet wird.

Diese Bachelorarbeit konzentriert sich auf die Herausforderung, ein Peer-to-Peer-Instant-Messaging-Protokoll zu konzipieren, welches eine neuartige, vertrauenswürdige und effiziente Infrastruktur bieten soll. Dabei liegt der Fokus nicht nur darauf, die Privatsphäre der Nutzer zu bewahren, sondern auch die Integrität und Authentizität der Kommunikation durch die Integration von Blockchain-Technologie zu gewährleisten. Das Hauptziel besteht darin, einen Prototypen für ein derartiges Protokoll zu entwickeln, der durch den Einsatz von Blockchain-Technologie die Sicherheit der Kommunikation verbessern soll.

Kapitel 2

Grundlagen

Diese Kapitel gibt einen Überblick über die Grundlagen, die für das Verständnis dieser Arbeit notwendig sind. Zunächst wird der Begriff *Instant-Messaging* definiert und es werden die beiden grundlegenden Arten von Instant-Messaging-Protokollen beschrieben. Darauf folgt eine kurze Vorstellung von bereits existierenden Instant-Messaging-Protokollen, die als Grundlage für die Entwicklung eines eigenen Protokolls dienen. Danach wird die Peer-to-Peer-Technologie beschrieben, mit deren Hilfe die Kommunikation zwischen den Benutzern eines dezentralen Instant-Messaging-Protokolls stattfindet. Anschließend werden Methoden zur Sicherung der Kommunikation beschrieben, die in der Regel in Instant-Messaging-Protokollen verwendet werden. Dies geschieht anhand der drei Sicherheitsziele Vertraulichkeit, Integrität und Authentizität. Als mögliches Vorbild für die Sicherung der Kommunikation wurde das *Signal*-Protokoll herangezogen. Abschließend wird die Blockchain-Technologie definiert und ihre Funktionsweise beschrieben.

2.1 Instant-Messaging

Instant-Messaging, was übersetzt *sofortige Nachrichtenübermittlung* bedeutet, bezeichnet eine Form der Kommunikation, bei der Nachrichten in Echtzeit zwischen zwei oder mehreren Personen über das Internet ausgetauscht werden können (Ayers, Brothers und Jansen 2014, S. 69). Diese Form der digitalen Kommunikation ermöglicht es Nutzern, sofortige Nachrichten, Bilder und andere Mediendateien auszutauschen. Instant-Messaging-Dienste reichen von plattformübergreifenden Anwendungen wie *WhatsApp*, *Signal* und *Telegram* bis hin zu spezialisierten Unternehmenslösungen wie *Slack* oder *Microsoft Teams* (Plett o. D.). Die Vielfalt an Funktionen in Instant-Messaging-Plattformen ist groß. Neben einfachen Textnachrichten können zum Beispiel Benutzer der Anwendung

WhatsApp Emojis, Aufkleber, GIFs (animierte Bilder) und Sprachnachrichten teilen, was die Kommunikation dynamisch und ausdrucksstark gestaltet (*WhatsApp Funktionen* o. D.).

Sicherheit und Datenschutz sind in der Welt des Instant Messaging von entscheidender Bedeutung. Verschlüsselungstechnologien werden verwendet, um die Vertraulichkeit der Nachrichten zu gewährleisten und die Privatsphäre der Nutzer zu schützen (Wang u. a. 2018, S. 13706).

2.1.1 Instant-Messaging-Anwendungen und -Protokolle

Durch die Verwendung von Instant-Messaging-Anwendungen können Benutzer Nachrichten in Echtzeit austauschen. Die Übertragung der Nachrichten erfolgt über ein Netzwerk (Ayers, Brothers und Jansen 2014, S. 69). Um die Kommunikation zu ermöglichen, müssen die Teilnehmer ein gemeinsames Protokoll verwenden. Ein Protokoll besteht aus einer Reihe von Regeln, die die Kommunikation zwischen Geräten (Computer oder auch Smartphones) ermöglichen. Damit beschreibt ein Instant-Messaging-Protokoll eine Reihe von Regeln, die die Kommunikation zwischen Benutzern einer Instant-Messaging-Anwendung erlauben (*Protokoll* o. D.). Im Jahr 2023 waren in Deutschland die folgenden Instant-Messaging-Anwendungen am beliebtesten: *WhatsApp*, *Facebook Messenger* und *Telegram* (Statista 2023). Diese Anwendungen verwenden unterschiedliche Ansätze, um die Kommunikation zu ermöglichen (Luntovskyy und Gütter 2020, S. 103).

2.1.2 Architekturmodelle

Instant-Messaging-Anwendungen können unterschiedliche Architekturmodelle verwenden, um die Kommunikation zwischen den Teilnehmern zu ermöglichen. Nachfolgend werden die Client-Server-Architektur und das Peer-to-Peer-Modell beschrieben.

Client-Server-Architektur

Bei dieser Art von Protokoll wird die Kommunikation über einen zentralen Server ermöglicht. Die Teilnehmer kommunizieren nicht direkt miteinander, sondern senden ihre Nachrichten an den Server, der sie dann an den Empfänger weiterleitet (Hanson 1999, S. 3). Einige Beispiele für Instant-Messaging-Anwendungen, die ein Protokoll mit zentralem Server verwenden, sind *WhatsApp* (Vanerio und Casas 2018), *Signal* (signalapp o. D.), *Telegram* (*Fragen und Antworten* o. D.) und *Threema* (*Where are the servers located?* o. D.).

Peer-to-Peer-Modell

Bei dieser Art von Protokoll kommunizieren die Teilnehmer direkt miteinander, ohne einen zentralen Server zu verwenden. Die Kommunikation erfolgt direkt zwischen den Teilnehmern (Mahlmann und Schindelhauer 2007, S. 6-8; Galuba und Girdzijauskas 2009). Einige Beispiele für Instant Messaging-Anwendungen, die ein Protokoll mit Peer-to-Peer-Modell verwenden, sind *Tox* (*Tox User FAQ* o. D.) und *Briar* (*Briar: How it works* o. D.).

2.1.3 Diskussion der wichtigsten Protokolle

Die folgenden Abschnitte beschreiben die wichtigsten Instant-Messaging-Protokolle, die für diese Arbeit relevant sind.

Signal

Das Signal-Protokoll ist ein Open-Source-Protokoll, das in der bekannten Signal-App verwendet wird. Es ist ein modernes Protokoll, das einen Fokus auf Sicherheit hat (*Signal Website* o. D.) und deshalb die Sicherheitsmaßnahmen des in dieser Arbeit entwickelten Protokolls maßgeblich beeinflusst hat. In Abschnitt 2.3.4 *Ende-zu-Ende-Verschlüsselung* wird der Sicherheitsaspekt des Signal-Protokolls genauer beschrieben. Ein Punkt, in dem sich die beiden Protokolle unterscheiden, ist die Verwendung von zentralen Servern. Das Signal-Protokoll setzt auf eine Client-Server-Architektur (signalapp o. D.), das Protokoll dieser Arbeit hingegen auf ein Peer-to-Peer-Modell.

Extensible Messaging and Presence Protocol (XMPP)

XMPP ist ein Open-Source-Protokoll, das für die Echtzeitkommunikation zwischen zwei Teilnehmern entwickelt wurde. Es ist ein dezentrales Protokoll, das auf dem Peer-to-Peer-Modell basiert (Saint-Andre 2011). Durch Erweiterung des Protokolls, die durch die *XMPP Standards Foundation* zugelassen werden müssen, können weitere Funktionen hinzugefügt werden (*XMPP Specifications* o. D.). XMPP findet in vielen Instant-Messaging-Anwendungen Verwendung, wie zum Beispiel *WhatsApp*, *Zoom* und *Jitsi* (*XMPP Instant Messaging* o. D.). Da aus der Spezifikation hervorgeht, dass die Verwendung von Servern ein fester Bestandteil ist (Saint-Andre 2011), wurde XMPP nicht als Referenz für das in dieser Arbeit entwickelte Protokoll verwendet. Der Fokus dieser Arbeit liegt auf einem dezentralen Protokoll, das ohne zentrale Server auskommt und nur in Ausnahmefällen auf Server zurückgreift.

Tox

Das Tox-Protokoll ist ebenfalls ein Open-Source-Protokoll, das für die Echtzeitkommunikation zwischen Teilnehmern entwickelt wurde (siehe 1 *Hintergrund und Motivation*). Es ermöglicht die Kommunikation zwischen zwei oder mehreren Teilnehmern und basiert auf dem Peer-to-Peer-Modell (*Tox User FAQ* o. D.). Tox wurde als Reaktion auf die Enthüllungen von Edward Snowden entwickelt, um eine sichere und dezentrale Alternative zu den bestehenden Instant-Messaging-Anwendungen zu schaffen (*Tox - The Tox Project* o. D.). Da Tox hauptsächlich ohne zentrale Server auskommt, wurde es als Referenz für das in dieser Arbeit entwickelte Protokoll verwendet.

2.2 Peer-to-Peer-Technologie

Peer-to-Peer-Technologien können in zwei Kategorien unterteilt werden: *Peer-to-Peer-Anwendungen* und *Peer-to-Peer-Infrastrukturen* (Khatibi und Sharifi 2020, S. 730).

Die Kategorie der Peer-to-Peer-Anwendungen umfasst den Dienst der *Inhaltsverteilung*, bei dem die Teilnehmer Inhalte wie Musik, Videos oder andere Dateien direkt untereinander austauschen (Khatibi und Sharifi 2020, S. 730–731).

Daher werden viele Peer-to-Peer schnell mit Filesharing in Verbindung bringen, da diese Technologie in der Vergangenheit vor allem dafür genutzt wurde. Das bekannteste Beispiel ist das Filesharing-Netzwerk *Napster*, das 1999 von Shawn „Napster“ Fanning entwickelt wurde. *Napster* war das erste weit verbreitete Filesharing-Netzwerk, das auf Peer-to-Peer-Technologie basierte. Es ermöglichte den Austausch von Musikdateien zwischen den Teilnehmern. Die Musikdateien wurden dabei auf den Computern der Teilnehmer gespeichert und konnten von anderen Teilnehmern heruntergeladen werden. Da diese Art des Datenaustauschs oftmals illegal war, wurde Napster 2001 aufgrund von Urheberrechtsverletzungen abgeschaltet (Mahlmann und Schindelhauer 2007, S. 55-57).

Die zweite Kategorie ist die Peer-to-Peer-Infrastruktur, welche die Peer-to-Peer-Netzwerke umfasst, die für die Kommunikation zwischen den Teilnehmern des Netzwerks verwendet werden (Khatibi und Sharifi 2020, S. 730-731). Hier sind alle Teilnehmer und fungieren sowohl als Client als auch als Server. Es gibt keine zentrale Instanz, die die Kommunikation steuert. Die Teilnehmer sind direkt miteinander verbunden und können Nachrichten oder andere Daten direkt austauschen.

Diese Art von Peer-to-Peer-Netzwerken wird in dieser Arbeit behandelt.

Im Bereich des Instant Messaging stellt das Peer-to-Peer-Modell eine dezentrale Struktur dar, die die Kommunikation zwischen den Nutzern eines Instant-Messaging-Dienstes

ermöglicht. Im Gegensatz zum Client-Server-Ansatz, bei dem ein zentraler Server die Kommunikation steuert, ermöglicht das Peer-to-Peer-Netzwerk eine direkte Kommunikation zwischen den Teilnehmern. Beide Modelle haben ihre Vor- und Nachteile. Während das Client-Server-Modell eine zentrale Instanz erfordert, um die Kommunikation zu verwalten, ist das Peer-to-Peer-Netzwerk dezentralisiert und benötigt eine solche Instanz nicht. Die Implementierung und Wartung eines Client-Server-Modells sind im Vergleich zu einem komplexeren und aufwendigeren Peer-to-Peer-Netzwerk einfacher. Das Client-Server-Modell ist weniger flexibel, da es von einer zentralen Instanz abhängt, während das Peer-to-Peer-Netzwerk aufgrund des Fehlens dieser Instanz flexibler ist. Ein weiterer Unterschied betrifft die Skalierbarkeit: Das Client-Server-Modell ist durch die Kapazität des Servers begrenzt, während das Peer-to-Peer-Netzwerk auf die Kapazität der Teilnehmer zurückgreift, was seine Skalierbarkeit verbessert. Hinsichtlich der Sicherheit ist das Client-Server-Modell weniger robust, da es auf eine zentrale Instanz angewiesen ist, während das Peer-to-Peer-Netzwerk als sicherer gilt, da es ohne eine solche Abhängigkeit auskommt (Mahlmann und Schindelhauer 2007, S. 6-8).

2.2.1 Typen von Peer-to-Peer-Netzwerken

Nicht jedes Peer-to-Peer-Netzwerk ist gleich aufgebaut. Es gibt verschiedene Typen von Peer-to-Peer-Netzwerken, die sich in ihrer Struktur und Funktionsweise unterscheiden. Abbildung 2.1 zeigt die zwei Haupttypen von Peer-to-Peer-Netzwerken: unstrukturierte und strukturierte Netzwerke (Luntovskyy und Gütter 2020, S. 362-363).

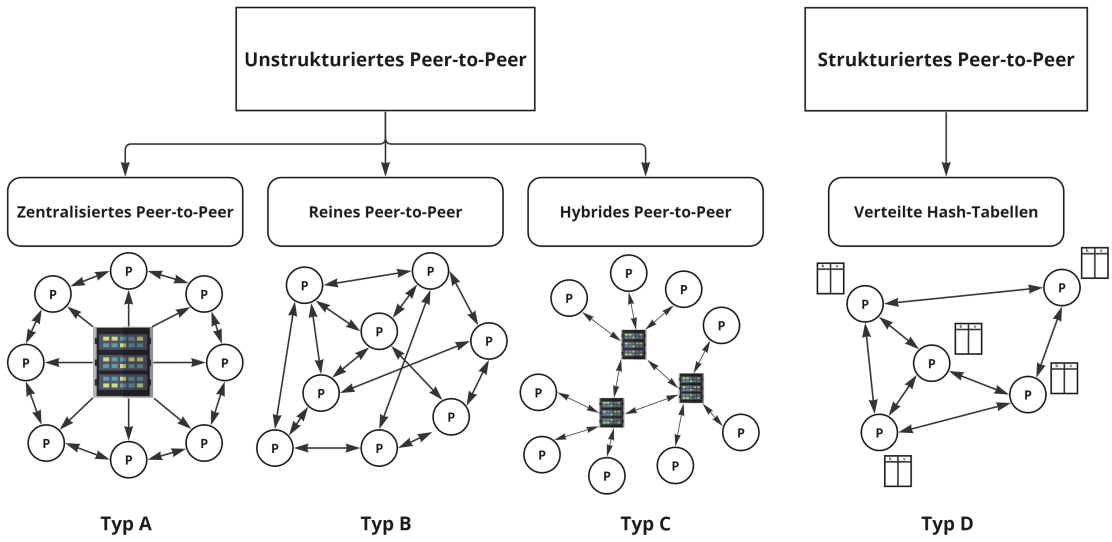

Abbildung 2.1: Typen von Peer-to-Peer-Netzwerken (in Anlehnung an Luntovskyy und Gütter 2020, S. 363)

Unstrukturierte und strukturierte Peer-to-Peer-Netzwerke sind unterschiedliche Ansätze zur Organisation von Knoten (engl. *Nodes*) und Ressourcen in dezentralen Netzwerken.

Unstrukturierte Netzwerke sind charakterisiert durch ihre fehlende explizite Organisationsstruktur, was eine einfache Konnektivität ermöglicht. *Typ A* in Abbildung 2.1 zeigt ein zentralisiertes Netzwerk, was bedeutet, dass alle Teilnehmer mit einem zentralen Server verbunden sind. Als Beispiel für diese Form des Peer-to-Peer dient *Napster*. Bei *Napster* gab es mehrere Server, die die Dateien der Teilnehmer indizierten. Die Teilnehmer konnten Dateien von anderen Teilnehmern herunterladen, indem sie eine Anfrage an einen der Server stellten, der dann die IP-Adresse des Teilnehmers zurückgab, der die gesuchte Datei zur Verfügung stellte (Saroiu, Gummadi und Gribble 2003, S. 171). Diese Form ermöglicht eine schnelle und effiziente Suche nach Ressourcen, da die Ressourcen zentral verwaltet werden, aber die Abhängigkeit von einem zentralen Server macht das Netzwerk nicht skalierbar und anfällig für Ausfälle (Khatibi und Sharifi 2020, S. 732). Bei *Typ B* handelt es sich um ein reines Peer-to-Peer-Netzwerk, bei dem die Teilnehmer direkt miteinander verbunden sind und jeder sowohl als Client als auch als Server fungiert (Khatibi und Sharifi 2020, S. 732). Ein Beispiel für diese Form des Peer-to-Peer ist *Gnutella*. Bei *Gnutella* gab es keine zentrale Instanz, die die Ressourcen der Teilnehmer indizierte. Die Suche nach Ressourcen oder Informationen erfolgt durch Broadcasts oder zufällige Weiterleitungen, was jedoch zu ineffizienten Suchprozessen führen kann, da keine klare Routing-Struktur vorhanden ist (Saroiu, Gummadi und Gribble 2003, S. 171). Beim dritten und letzten Typ (*Typ C*) der unstrukturierten Netzwerke handelt es sich um ein hybrides Peer-to-Peer-Netzwerk, das Elemente aus den beiden anderen Typen kombiniert. In einem hybriden Netzwerk gibt es besondere Knoten, die die Funktionen eines Servers, wie beispielsweise Indexierung der Ressourcen, für eine bestimmte Gruppe von Teilnehmern übernehmen. Diese Knoten werden als Superknoten (engl. *Super Nodes*) bezeichnet. Die Super Nodes selbst sind untereinander dezentralisiert miteinander verbunden. Ein Beispiel für diese Form des Peer-to-Peer ist *Gnutella2* (Khatibi und Sharifi 2020, S. 732).

Strukturierte Peer-to-Peer-Netzwerke hingegen weisen klare Regeln und Algorithmen zur Organisation der Knoten auf. Diese Netzwerke verfügen über eine explizite Organisationsstruktur, sei es eine Ringstruktur, k-bucket basierte Systeme oder andere, die es ermöglichen, effizientes Routing und eine optimierte Ressourcenverwaltung zu erreichen. Durch diese klar definierte Struktur sind strukturierte Netzwerke oft stabiler und bieten eine effizientere Ressourcenlokalisierung im Vergleich zu ihren unstrukturierten

Gegenstücken. Allerdings kann diese Stabilität auf Kosten von Flexibilität und Anpassungsfähigkeit gehen, da Änderungen in der Netzwerktopologie oder hohe Dynamik der Knoten schwerer zu handhaben sind (Vu, Lupu und Ooi 2010, S. 40).

2.2.2 Problemstellung und mögliche Lösungen

Leider bringen Peer-to-Peer-Netzwerke auch einige Probleme mit sich. Eines der Probleme stellen die *Network Address Translators* (kurz: *NATs*) dar. NATs sind dafür zuständig, private IP-Adressen in öffentliche IP-Adressen umzuwandeln und umgekehrt. Sie werden in Routern oder Gateways eingesetzt und dienen dazu, den Zugang von Geräten im lokalen Netzwerk, welche private IP-Adressen verwenden, zum Internet zu ermöglichen, indem sie den Datenverkehr zwischen dem lokalen Netzwerk und dem externen Netzwerk, wie dem Internet, verwalten. Peer-to-Peer Verbindungen stoßen bei Network Address Translators oft auf Probleme, was daran liegt, dass NATs normalerweise nicht erlauben, dass externe Geräte direkt mit internen Geräten kommunizieren. Zudem werden Ports dynamisch für ausgehenden Traffic zugewiesen, was das Weiterleiten eingehender Verbindungen erschwert. Symmetrische NATs verschärfen dieses Problem, da sie für ausgehende Verbindungen eine eindeutige Kombination von IP-Adresse und Port verwenden, die sich bei jeder neuen Verbindung ändert (Holdrege und Srisuresh 1999, S. 1-9).

NATs stellen also eine große Herausforderung für Peer-to-Peer-Netzwerke dar, da sie die direkte Kommunikation zwischen den Teilnehmern erschweren, wenn sie sich hinter einem NAT befinden. Um dieses Problem zu lösen, gibt es verschiedene Lösungsansätze. Einer davon ist das *Relaying*. Das Verb *to relay* kann mit *weiterleiten* oder *weitergeben* übersetzt werden. Beim Relaying wird also ein Server als Vermittler zwischen den Teilnehmern verwendet, der die Nachrichten zwischen den Teilnehmern weiterleitet. Damit sich die Teilnehmer mit dem Server verbinden können, muss dieser eine öffentliche IP-Adresse haben, die jedem Teilnehmer bekannt ist. Dieser Ansatz ist einfach zu implementieren, erfordert aber einen zentralen Server, der diese Aufgabe übernimmt. Zudem ist er nicht sehr effizient, da der Server die Nachrichten zwischen den Teilnehmern weiterleiten muss, was zu einer höheren Latenz führen kann. Dazu kommt, dass ein solcher Server einen *Single Point of Failure* oder zu Deutsch *einzelner Ausfallpunkt* darstellt, da die Kommunikation zwischen den Teilnehmern unterbrochen wird, wenn der Server ausfällt.

Ein weiterer Ansatz ist die *Connection Reversal*. Bei der Connection Reversal wird ein *Rendezvous-Server* verwendet, um eine Verbindung zwischen den Teilnehmern herzustellen. Der Teilnehmer, der sich im öffentliche Netzwerk befindet, möchte eine Verbindung

mit dem Teilnehmer herstellen, der sich hinter dem NAT befindet. Der Aufbau einer direkten Verbindung ist fehlgeschlagen, weshalb der Teilnehmer eine Verbindung mit dem Rendezvous-Server herstellt. Mit Hilfe des Rendezvous-Servers teilt der Teilnehmer im öffentlichen Netzwerk dem Teilnehmer hinter dem NAT seine öffentliche IP-Adresse und Port mit, sodass dieser den Verbindungsaufbau einleitet. Bei dieser Technik darf sich jedoch nur einer der Teilnehmer hinter einem NAT befinden.

Hole Punching beschreibt einen weiteren Lösungsansatz. Zwei Geräte, die eine direkte Verbindung miteinander aufbauen möchten, initiieren gleichzeitig eine Verbindung zu einem Server, der sich außerhalb des NATs befindet. Der Server sammelt die IP-Adressen und Ports der beiden Geräte und leitet diese an die jeweils andere Partei weiter. Die beiden Geräte versuchen dann, eine Verbindung zueinander herzustellen, indem sie gleichzeitig Datenpakete an die IP-Adresse und den Port des anderen Geräts senden. Dabei wird versucht, das NAT dazu zu bringen, die Verbindung zu öffnen, indem es die ankommenden Pakete als Antwort auf die ausgehenden Pakete erkennt. Wenn dies gelingt, wird ein temporäres Loch im NAT geöffnet, das es den Geräten ermöglicht, direkt miteinander zu kommunizieren. Diese Technik erfordert eine präzise Koordination und die Fähigkeit der beiden Geräte zur gleichen Zeit Datenpakete zu senden und zu empfangen. Zudem ist es nicht immer möglich, ein temporäres Loch im NAT zu öffnen, da es von der Implementierung des NATs abhängt.

Eine Abwandlung vom Hole Punching ist die *Port Number Prediction*. Hierbei wird versucht, die Portnummer vorherzusagen, die das NAT für die Verbindung verwenden wird. Durch Beobachtung und Analyse vorheriger Verbindungen wird versucht Muster oder Trends in der Art und Weise zu erkennen, wie Portnummern zugewiesen werden. Dies könnte auf bestimmte Algorithmen oder Verhaltensweisen des Systems hinweisen, woraus dann die Portnummer vorhergesagt werden kann. Diese Technik ist jedoch nicht immer zuverlässig, da es rein auf Annahmen basiert und das Risiko besteht, dass sich das Portzuweisungsmuster jederzeit ändern könnte (Ford, Kegel und Srisuresh 2008, S. 7-21).

Um diese Problematik von Peer-to-Peer-Netzwerken zu lösen, können verschiedene Protokolle zum Einsatz kommen. Eines dieser Protokolle ist *STUN* (kurz für *Session Traversal Utilities for NAT*). STUN ist ein Netzwerkprotokoll, das es Geräten, die sich hinter einem NAT befinden, ermöglicht, ihre öffentliche IP-Adresse und Port zu ermitteln. Es bietet an sich keine Möglichkeit für eine Umgehung des NATs, sondern ist dafür gedacht, als eines von mehreren Werkzeugen verwendet zu werden, um ein NAT zu umgehen. Mittels STUN lässt sich nur ermitteln, ob sich ein Gerät hinter einem NAT befindet und wenn dies zutrifft, welche IP-Adresse und Port es verwendet (Petit-Huguenin u. a.

2020, S. 4).

TURN (kurz für *Traversal Using Relays around NAT*) ist ein weiteres Netzwerkprotokoll, das im Zusammenhang mit NATs verwendet werden kann. Aus der Spezifikation ist zu entnehmen, dass TURN ein Protokoll ist, das es Geräten, die sich hinter einem NAT befinden, ermöglicht, eine Verbindung zu einem anderen Gerät herzustellen, indem es einen Server als Vermittler verwendet (siehe weiter oben in 2.2.2 *Problemstellung und mögliche Lösungen*). Das funktioniert auch, wenn sich beide Geräte hinter einem NAT befinden (Reddy u. a. 2020, S. 7).

ICE (kurz für *Interactive Connectivity Establishment*) ist ein Framework, das mehrere Techniken kombiniert, um eine Verbindung zwischen zwei Endpunkten herzustellen, die sich hinter NATs befinden. Es verwendet sowohl STUN als auch TURN, um die öffentliche IP-Adresse und Port eines Geräts zu ermitteln und den Datenverkehr über einen Server zu leiten (Keränen, Holmberg und Rosenberg 2018, S. 6).

2.2.3 Overlay-Netzwerke

Peer-to-Peer-Teilnehmer sind meistens nicht direkt miteinander verbunden, sondern zum Beispiel über das Internet. Deshalb werden Peer-to-Peer-Netzwerke auch als Overlay-Netzwerke bezeichnet. Ein Overlay-Netzwerk ist ein virtuelles Netzwerk, das, wie in Abbildung 2.2 (*Peer-to-Peer-Overlay-Netzwerk mit darunterliegendem physischen Netzwerk (Kunzmann 2009)*) zu sehen ist, über ein physisches Netzwerk gelegt wird.

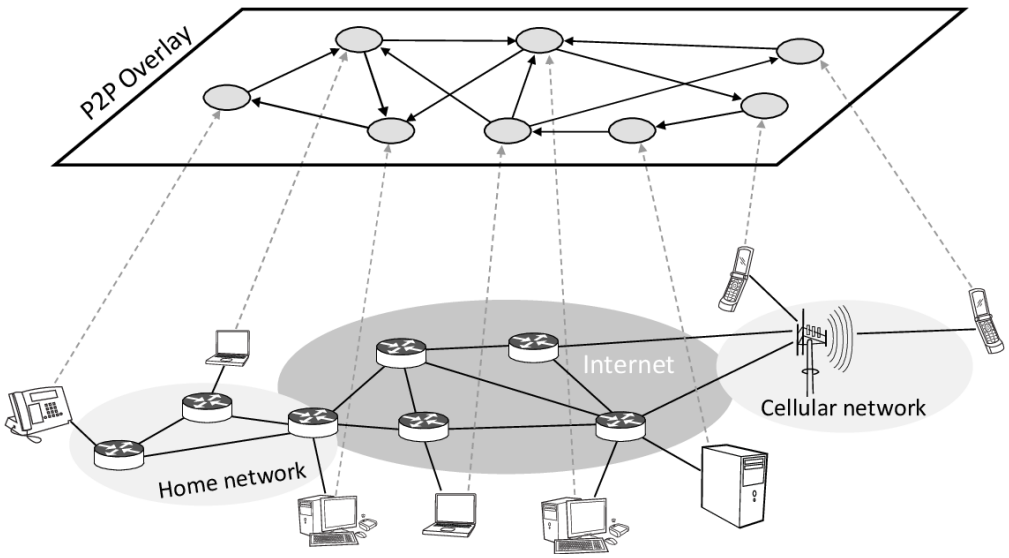

Abbildung 2.2: Peer-to-Peer-Overlay-Netzwerk mit darunterliegendem physischen Netzwerk (Kunzmann 2009)

In diesem Fall ist das physische Netzwerk das Internet. Das Overlay-Netzwerk ist eine logische Struktur, die es ermöglicht, die Kommunikation zwischen den Teilnehmern zu organisieren. Es besteht aus einer Reihe von Knoten, die über eine logische Verbindung miteinander verbunden sind. Die Verbindungen zwischen den Knoten werden durch Routing-Algorithmen verwaltet (Lua u. a. 2005).

Beispiele für diese Routing-Algorithmen sind *Kademlia*, *Chord* und *Pastry*. Diese drei Algorithmen verwenden sogenannte *Distributed Hash Tables* (zu Deutsch: *Verteilte Hashtabellen*), um die Knoten zu verwalten. Distributed Hash Tables (kurz: *DHTs*) sind verteilte Datenstrukturen, die in Peer-to-Peer-Netzwerken verwendet werden, um effizient Schlüssel-Wert-Paare zu speichern und abzurufen. Anders als herkömmliche zentralisierte Datenbanken oder Speicherlösungen, welche die Daten auf einem zentralen Server speichern, benötigen DHTs keinen solchen Server zur Speicherung oder Verwaltung der Daten. Sie funktionieren auf Basis von Hashfunktionen (siehe 2.3.2 *Integrität*), die einen Schlüssel in einen eindeutigen Hash umwandeln. Diese Hashes dienen als Adressen, um zu bestimmen, wo die entsprechenden Daten im Netzwerk gespeichert sind. Die Daten werden über verschiedene Peers im Netzwerk verteilt, wobei jeder Peer nur einen Teil der Daten basierend auf seinem Verantwortungsbereich speichert. Diese Algorithmen ermöglichen es, Peers im Netzwerk zu finden, die für die Speicherung oder Abfrage von Daten zuständig sind, selbst wenn sich die Netzwerktopologie ständig verändert. Ein großer Vorteil von DHTs ist ihre Skalierbarkeit. Sie können mit der Netzwerkgröße wachsen, ohne an Effizienz zu verlieren. Neue Peers können nahtlos hinzugefügt werden, und die Struktur der DHT passt sich dynamisch an Veränderungen im Netzwerk an (Stoica u. a. 2001; Rowstron und Druschel 2001; Maymounkov und Mazières 2002; Balakrishnan u. a. 2003, S. 43-46).

2.2.4 Kademlia vs. Chord vs. Pastry

Kademlia ist ein Routing-Algorithmus, der auf einer k-Bucket-Struktur basiert. Das k in k-Bucket steht für die Anzahl der Knoten, die in einem Bucket gespeichert werden können. Bei einem niedrigen Wert für k reduziert den Aufwand, der benötigt wird, um eine Routing-Tabelle zu verwalten, aber erhöht das Risiko für das Versagen eines Knotens (engl. *Node Failure*). Bei einem hohen Wert für k ist es genau umgekehrt. Routing-Tabellen mit mehreren Buckets sind in der Lage, mehr (redundante) Knoten zu speichern, was das Routing sicherer vor Fehlern macht. Allerdings kostet die Verwaltung von mehr Buckets auch mehr Aufwand. Die Wahl eines geeigneten Wertes für k ist daher entscheidend für die Leistungsfähigkeit, Fehlertoleranz und Skalierbarkeit des

Netzwerks.Die gespeicherten Knoten werden in Buckets organisiert, wobei jeder Bucket für einen bestimmten Schlüsselbereich verantwortlich ist. Die Verbindungen zwischen den Knoten werden durch die XOR-Distanz der IDs definiert und sind asymmetrisch. Das bedeutet, dass ein Knoten A eine Verbindung zu einem anderen Knoten B haben kann, aber deshalb B keine Verbindung zu A haben muss. Bei der Suche nach einem bestimmten Schlüssel erfolgt das Routing durch die XOR-Entfernung, wodurch die nächsten Knoten für diesen Schlüssel gefunden werden. Dieses Verfahren ermöglicht eine logarithmische Anzahl von Schritten für die Suche und bietet eine robuste Struktur, die gut mit dynamischen Netzwerkänderungen umgehen kann (Maymounkov und Mazières 2002, S. 1-2).

Chord ist ebenfalls ein Routing-Algorithmus, basiert allerdings auf einer Ringstruktur. Die Knoten sind in einem Ring angeordnet und jeder Knoten ist für einen bestimmten Schlüsselbereich verantwortlich. Die Verbindungen zwischen den Knoten sind durch ihren Platz im Ring definiert, wobei jeder Knoten eine Verbindung zu seinem nächsten Nachbarn im Uhrzeigersinn hat. Bei der Suche nach einem bestimmten Schlüssel durchläuft eine Anfrage einen logarithmischen Pfad im Ring, wobei die Knoten auf dem Weg begrenzte Informationen über andere Knoten behalten, um Anfragen weiterzuleiten. Dieses Modell ist recht einfach und effizient für viele Anwendungsfälle, aber es könnte anfällig sein für Engpässe oder längere Suchzeiten, insbesondere wenn das Netzwerk dynamisch ist und sich die Konfiguration häufig ändert (Stoica u. a. 2001, S. 1-3).

Pastry ist ein weiterer Routing-Algorithmus, der auf einer eindimensionalen Struktur basiert. Die Knoten sind in einem eindimensionalen Adressraum angeordnet, wobei jeder Knoten für einen bestimmten Schlüsselbereich verantwortlich ist. Die Verbindungen zwischen den Knoten werden durch eine gemeinsame Präfix-Länge definiert, wobei jeder Knoten eine Verbindung zu seinem nächsten Nachbarn hat. Jeder Knoten besitzt drei Listen: eine *Routing-Tabelle*, eine *Nachbarn-Liste* und eine *Blatt-Liste*. Die Routing-Tabelle enthält Informationen über andere Knoten im Netzwerk, die für bestimmte Präfixe verantwortlich sind. Die Nachbarn-Liste enthält Informationen über die Knoten im Netzwerk, die am nächsten an diesem Knoten liegen. Die Blatt-Liste enthält Informationen über Knoten, die am nächsten und am weitesten von diesem Knoten entfernt sind. Dies hilft bei der Suche nach einem bestimmten Schlüssel, der nicht durch die Routing-Tabelle abgedeckt wird. Bei der Suche wird nach einem Knoten gesucht, der die größte Übereinstimmung im ID-Präfix besitzt. Dieser Knoten wird dann die Anfrage an den nächsten Knoten weiterleiten, der eine größere Übereinstimmung im ID-Präfix besitzt. Dieser Vorgang wird wiederholt, bis der Knoten gefunden wird, der für den Schlüssel verantwortlich ist. Dieses Verfahren ermöglicht eine logarithmische Anzahl von Schritten für die Suche und bietet eine gute Balance zwischen Effizienz und Skalierbarkeit (Rowstron

und Druschel 2001).

Insgesamt bieten alle drei Algorithmen Lösungen für die mögliche Anwendung in einem Instant-Messaging-Kontext. Je nach Anwendungsfall können sie unterschiedliche Vorteile bieten. Kademlia ist gut geeignet für große, dynamische Netzwerke, Chord für statischere Netzwerke und Pastry für mittelgroße Netzwerke mit moderater Dynamik. Für den Anwendungsfall des Instant-Messaging ist die effektive Bewältigung von *Churn* von entscheidender Bedeutung. Churn bezieht sich auf die häufigen Ein- und Austritte von Teilnehmern in einem Peer-to-Peer-Netzwerk. In einem Instant-Messaging Kontext bedeutet dies, dass Benutzer sich ständig anmelden oder abmelden. Ein Protokoll, das gut mit Churn umgehen kann, ist entscheidend, um eine zuverlässige und nahtlose Kommunikation zu gewährleisten. Das richtige Handling von Churn ist daher ein Schlüsselfaktor für die Leistungsfähigkeit und Stabilität eines Instant-Messaging-Protokolls (Peris, Hernández und Huedo 2015, S. 316-317).

2.2.5 Angriffe auf Peer-to-Peer-Netzwerke

Peer-to-Peer-Netzwerke sind anfällig für verschiedene Arten von Angriffen. Diese Angriffe können in zwei Kategorien unterteilt werden: *Angriffe auf die Integrität* und *Angriffe auf die Verfügbarkeit*.

Angriffe auf die Integrität

Bei Angriffen auf die Integrität geht es darum, die Integrität der Daten zu gefährden, die im Netzwerk gespeichert sind. Ein Beispiel für einen solchen Angriff ist der *Sybil-Angriff*. Bei diesem Angriff erstellt ein einzelner Angreifer mehrere Identitäten, um die Kontrolle über das Netzwerk zu erlangen (Douceur 2002, S. 251). Nach einer erfolgreichen Übernahme von Teilen des Netzwerks, besteht die Möglichkeit der Durchführung eines *Eclipse*-Angriffs (Prêtre 2005, S. 13-15). Bei einem *Eclipse*-Angriff wird versucht, einen Peer von allen anderen Peers im Netzwerk zu isolieren. Dies wird erreicht, indem bereits durch den vorhergegangenen Sybil-Angriff die Mehrheit der Peers kontrolliert wird. Der Angreifer kann dann die Verbindungen des Opfers zu anderen Peers im Netzwerk verhindern, indem er die Verbindungen zu anderen Peers kontrolliert (Prêtre 2005, S. 14).

Angriffe auf die Verfügbarkeit

Bei Angriffen auf die Verfügbarkeit geht es darum, die Verfügbarkeit des Netzwerks zu gefährden. Ein Beispiel für einen solchen Angriff ist der *Denial-of-Service-Angriff*. Mit

einem *Denial-of-Service*-Angriff (kurz: *DoS-Angriff*) wird versucht, die Verfügbarkeit eines Dienstes zu beeinträchtigen, indem die Ressourcen des Dienstes erschöpft werden, sodass diese ausfallen (Bicakci und Tavli 2009). In einem Peer-to-Peer-Netzwerk kann ein DoS-Angriff auf verschiedene Arten durchgeführt werden. Eine Möglichkeit ist es, einen Peer mit Anfragen zu überfluten, um ihn zu überlasten. Eine andere Möglichkeit ist es, einen Peer mit gefälschten Informationen zu überfluten, um ihn zu täuschen. Beide Methoden führen dazu, dass der Peer nicht mehr in der Lage ist, seine Aufgaben zu erfüllen, was zu einem Ausfall des Dienstes führt. Dieser Angriff wird noch effektiver, wenn er von mehreren Angreifern gleichzeitig durchgeführt wird. Dies wird als *Distributed Denial-of-Service*-Angriff (kurz: *DDoS-Angriff*) bezeichnet. Bei einem DDoS-Angriff können mehrere Peers gleichzeitig mit Anfragen überflutet werden, was es schwieriger macht, den Angriff zu stoppen (Prêtre 2005, S. 6). Aber auch ein erfolgreicher Sybil-Angriff kann dazu führen, dass die Verfügbarkeit des Netzwerks gefährdet wird. Wenn ein Angreifer die Mehrheit der Peers kontrolliert, kann er die Verbindungen zu anderen Peers kontrollieren. Die Kommunikation zwischen den Peers kann dann durch falsches Routing verlangsamt oder auch ganz verhindert werden (Prêtre 2005, S. 13).

2.3 Sicherheit

Wie aus der Abbildung in Abschnitt 2.2.3 *Overlay-Netzwerke* ersichtlich ist, ist das Overlay-Netzwerk die oberste Schicht des Peer-to-Peer-Netzwerks. Es sorgt dafür, dass sich die Teilnehmer des Netzwerks finden können. Die eigentliche Kommunikation, also das Senden und Empfangen von Nachrichten, erfolgt jedoch über das Internet. Da das Internet ein öffentliches Netzwerk ist, besteht die Gefahr, dass die Nachrichten abgefangen und mitgelesen werden können. Die Nachrichten müssen über Verteilerknoten oder Access Points übertragen werden, die nicht vertrauenswürdig sind (Wong 2023, S. 235). Um ein Mitlesen oder Verändern der Nachrichten zu verhindern, muss die Kommunikation abgesichert werden.

Mittels Kryptografie kann die Vertraulichkeit, Integrität und Authentizität der Kommunikation gewährleistet werden (Hellmann 2023, S. 7).

2.3.1 Vertraulichkeit

Die Vertraulichkeit der Kommunikation wird durch Verschlüsselung gewährleistet. Man unterschiedet zwei Formen der Kryptografie: *symmetrische* und *asymmetrische* Kryptografie. Bei der *symmetrischen* Kryptografie wird ein und derselbe Schlüssel zum Ver-

schlüsseln und Entschlüsseln der Nachricht verwendet. Der Sender der Nachricht verschlüsselt die Nachricht mit einem Schlüssel und sendet die nun verschlüsselte Nachricht an den Empfänger. Der Empfänger kann die Nachricht mit dem gleichen Schlüssel entschlüsseln. Dadurch kann ein Angreifer, der die Nachricht abfängt, diese nicht entschlüsseln, da er den Schlüssel nicht kennt. Abbildung 2.3 zeigt den Ablauf der symmetrischen Verschlüsselung.

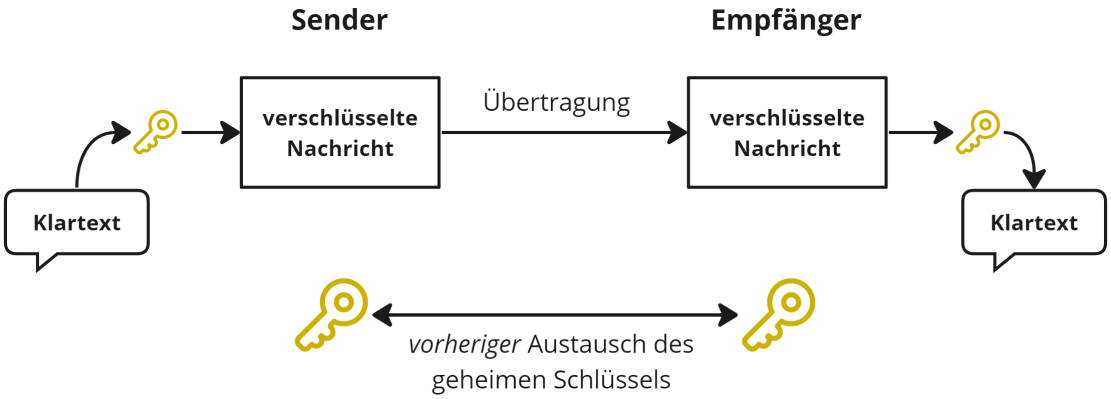

Abbildung 2.3: Symmetrische Verschlüsselung (in Anlehnung an *Symmetrische Kryptografie (Verschlüsselung)* o. D.)

Das Problem bei der symmetrischen Kryptografie ist, dass der Schlüssel zu Beginn der Kommunikation vom Sender an den Empfänger gelangen muss. Dies stellt eine Herausforderung dar, wenn Sender und Empfänger sich noch nicht kennen und noch nie zuvor miteinander kommuniziert haben oder noch nicht über andere Wege einen Schlüssel ausgetauscht haben. Sollte der Schlüssel bei der Übertragung über einen unsicheren Kanal abgefangen werden, kann der Angreifer die Kommunikation entschlüsseln und somit mitlesen (Diffie und Hellman 1976, S. 644; Wong 2023, S. 5-8).

In diesem Fall kann eine Schlüsselvereinbarung verwendet werden, um einen gemeinsamen Schlüssel zu erhalten. Bei der Schlüsselvereinbarung wird ein Schlüssel zwischen zwei Parteien vereinbart, ohne dass dieser über einen unsicheren Kanal übertragen werden muss (Wong 2023, S. 102). Abbildung 2.4 zeigt den Ablauf der Schlüsselvereinbarung.

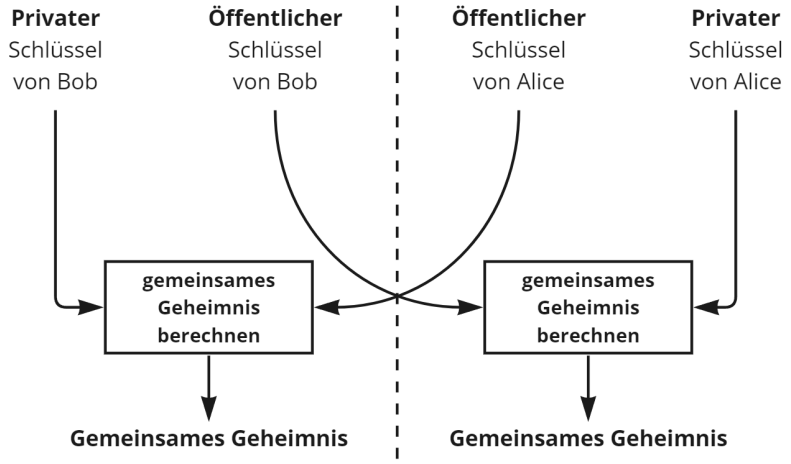

Abbildung 2.4: Schlüsselvereinbarung (in Anlehnung an Wong 2023, S. 102)

Beide Teilnehmer generieren einen privaten Schlüssel und einen öffentlichen Schlüssel. Durch die Kombination des öffentlichen Schlüssels des anderen Teilnehmers und des eigenen privaten Schlüssels wird ein gemeinsames Geheimnis berechnet. Dieses gemeinsame Geheimnis kann dann für die symmetrische Verschlüsselung verwendet werden, da dadurch beide Teilnehmer den gleichen Schlüssel besitzen (Wong 2023, S. 102).

Bei der *asymmetrische* Kryptografie (auch *Public-Key-Kryptografie* genannt) wird anstatt nur eines Schlüssels ein Schlüsselpaar generiert, das aus einem öffentlichen und einem privaten Schlüssel besteht. Der öffentliche Schlüssel des Empfängers wird zum Verschlüsseln der Nachrichten verwendet und zum Entschlüsseln der Nachrichten wird der private Schlüssel des Empfängers verwendet. Der öffentliche Schlüssel des Empfängers kann von jedem verwendet werden, um Nachrichten an den Empfänger zu verschlüsseln. Nur der Empfänger kann die Nachrichten entschlüsseln, da nur er den privaten Schlüssel besitzt. Der Ablauf der asymmetrischen Verschlüsselung ist in Abbildung 2.5 dargestellt.

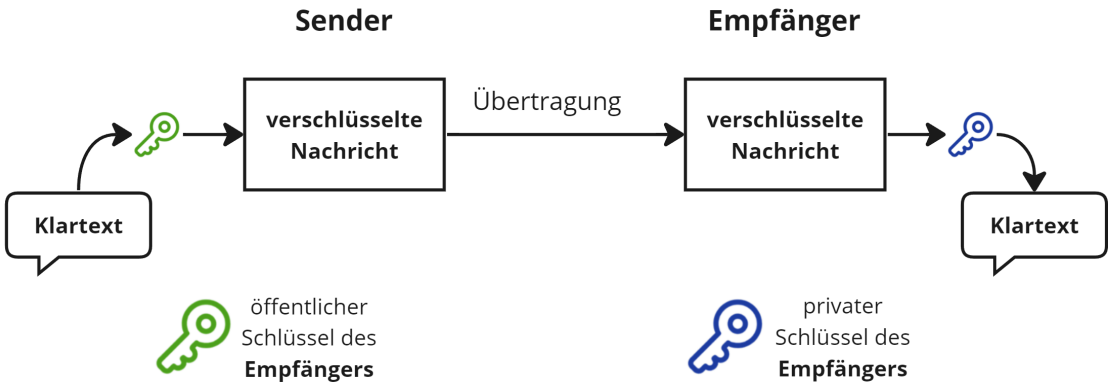

Abbildung 2.5: Asymmetrische Verschlüsselung (in Anlehnung an *Asymmetrische Kryptografie (Verschlüsselung)* o. D.)

Es ist außerdem nicht möglich, Nachrichten, die mit dem öffentlichen Schlüssel verschlüsselt wurden, mit diesem auch wieder zu entschlüsseln (*Asymmetrische Kryptografie (Verschlüsselung)* o. D.).

Diese kryptografische Verfahren können kombiniert werden, um die Vorteile von diesen zu nutzen. Der Schlüsselaustausch wird mit der asymmetrischen Verschlüsselung durchgeführt und die eigentliche Kommunikation wird mit der symmetrischen Verschlüsselung durchgeführt.

2.3.2 Integrität

Um die Integrität der Kommunikation zu gewährleisten, wird Hashing verwendet. Das Hashing von Daten erfordert die Verwendung einer *Hash-Funktion*. Hash-Funktionen sind Funktionen, die eine Eingabe beliebiger Länge in eine Ausgabe fester Länge umwandeln. Dabei ist es wichtig, dass die Hash-Funktion zwei Eigenschaften erfüllt: *Einwegfunktion* und *Kollisionsresistenz*. Eine Hash-Funktion erfüllt die Eigenschaft der Einwegfunktion, wenn es nicht möglich ist, von der Ausgabe auf die Eingabe zu schließen. Das bedeutet, dass es nicht möglich ist, aus dem Hash-Wert die ursprünglichen Daten zu rekonstruieren. Die Eigenschaft der Kollisionsresistenz ist erfüllt, wenn es nicht möglich ist, zwei verschiedene Eingaben zu finden, die auf den gleichen Hash-Wert abgebildet werden (Brünnler 2018, S. 12-13; Fill und Meier 2020, S. 6).

Das Ergebnis der Hash-Funktion ist eine Zeichenkette, die aus einer festen Anzahl an Zeichen besteht und als *Hash-Wert* oder auch *Digest* bezeichnet wird. Der Hash-Wert ist ein eindeutiger Fingerabdruck der Daten, die in die Hash-Funktion eingegeben wurden. Sollte sich also der Inhalt der Daten ändern, ändert sich auch der Hash-Wert (Wong 2023, S. 29-32).

2.3.3 Authentizität

In Kombination mit einer sogenannten Signatur kann zusätzlich zur Integrität auch die Authentizität einer Nachricht gewährleistet werden. Abbildung 2.6 zeigt, wie eine Nachricht digital signiert wird.

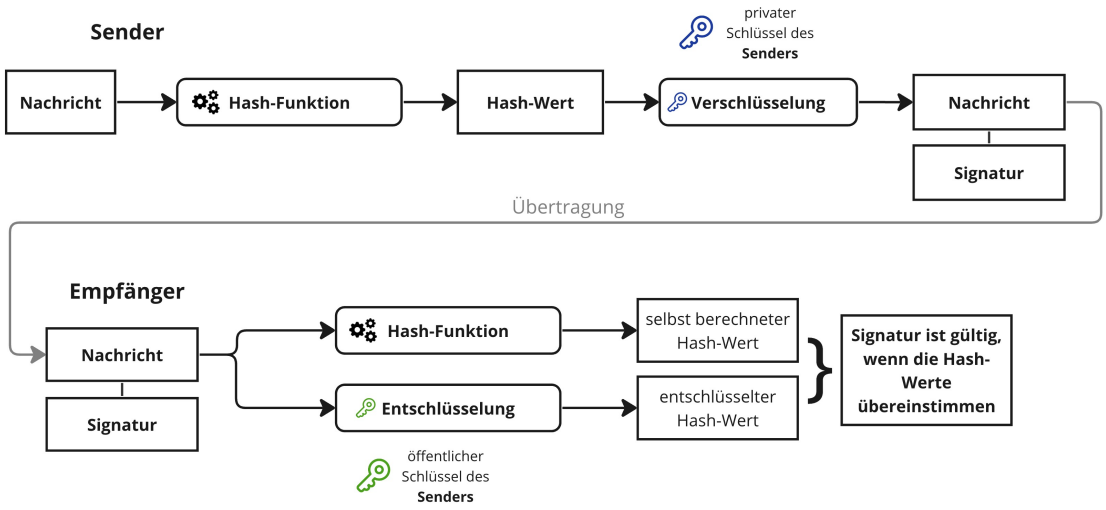

Abbildung 2.6: Signieren einer Nachricht (in Anlehnung an *Was sind digitale Signaturen?* o. D.)

Der Sender berechnet den Hash-Wert der Nachricht, verschlüsselt diesen mit seinem privaten Schlüssel. Das Ergebnis ist die digitale Signatur, welche an die Nachricht angehängt wird. Der Empfänger braucht den öffentlichen Schlüssel des Senders, um die Signatur zu entschlüsseln. Dafür gibt es verschiedene Möglichkeiten. Eine Möglichkeit ist, dass der Sender den öffentlichen Schlüssel dem Empfänger vorher über einen sicheren Kanal übermittelt. Eine andere Möglichkeit ist, dass der öffentliche Schlüssel des Senders in einem öffentlichen Schlüsselverzeichnis gespeichert ist. Wenn der Empfänger den öffentlichen Schlüssel des Senders besitzt, kann er die Signatur entschlüsseln. Gleichzeitig berechnet der Empfänger den Hash-Wert der Nachricht. Wenn der berechnete Hash-Wert mit dem entschlüsselten Hash-Wert übereinstimmt, kann der Empfänger einerseits sicher sein, dass die Nachricht nicht verändert wurde und somit die Integrität der Nachricht gewährleistet ist und andererseits, dass die Nachricht vom Sender stammt und somit die Authentizität der Nachricht gewährleistet ist. Falls der berechnete Hash-Wert nicht mit dem entschlüsselten Hash-Wert übereinstimmt, wurde die Nachricht entweder verändert oder die Nachricht stammt nicht vom erwarteten Sender (Hellmann 2023, S. 73-78).

2.3.4 Ende-zu-Ende-Verschlüsselung

Die behandelten Sicherheitsmechanismen können zu einer sogenannten Ende-zu-Ende-Verschlüsselung kombiniert werden. Diese erlaubt es, dass nur der Sender und der Empfänger die Nachrichten lesen können. Eine moderne Variante der Ende-zu-Ende-Verschlüs-

selung ist das Signal-Protokoll. Dies wird auch in der Signal-App verwendet (2.1.3 *Signal*). Im Folgenden wird das Signal-Protokoll genauer beschrieben.

Zu Beginn einer Sitzung wird ein gemeinsamer Schlüssel zwischen den Teilnehmern vereinbart. Das Signal-Protokoll verwendet hierfür ein spezielles Verfahren, das sich *Extended Triple Diffie-Hellman* oder auch kurz *X3DH* nennt. Dieses kombiniert mehrere Schlüsselaustauschaktionen, und ist dadurch in der Lage, einen gemeinsamen Schlüssel zu berechnen, ohne dass der Kommunikationspartner direkt erreichbar sein muss. Dazu werden mehrere flüchtige Schlüssel auf einem Server gespeichert. Dies dient dazu, dass eine Kompromittierung dieser Schlüssel nicht die Sicherheit der Kommunikation gefährdet, da diese nur für eine begrenzte Zeit gültig sind und für jede Sitzung neu generiert werden (Wong 2023, S. 249-252).

Da solch eine Sitzung sehr lange dauern kann, generiert das Signal-Protokoll auch für jede Nachricht einen neuen Schlüssel. Hierfür wird der sogenannte *Double Ratchet Algorithmus* verwendet. Dieser bildet den Kern des Signal-Protokolls. Eine *Ratchet* (zu Deutsch: *Ratsche*) ist ein Werkzeug, das nur in eine Richtung gedreht werden kann. Diese Eigenschaft wird auf die verwendeten Schlüssel angewendet. Dadurch ist es nicht möglich, vorherige Schlüssel zu berechnen, was auch als *Forward Secrecy* bezeichnet wird. Der Algorithmus verwendet zwei Ratchet-Schritte. Der erste Ratchet-Schritt verwendet einen symmetrischen Schlüssel. Basierend auf dem vorher, durch X3DH vereinbarten, gemeinsamen Schlüssel, generieren die Kommunikationspartner jeweils einen Empfangs- und einen Sendeschlüssel. Jede gesendete Nachricht wird dem Sendeschlüssel verschlüsselt und jede empfangene Nachricht wird mit dem Empfangsschlüssel entschlüsselt. Nach jeder Nachricht werden die Schlüssel durch eine Schlüsselableitungsfunktion aktualisiert. Der zweite Ratchet-Schritt ist ein asymmetrischer Schlüsselaustausch, den die Kommunikationspartner periodisch austauschen (Wong 2023, S. 252-257).

Vereinfacht kann der Double Ratchet Algorithmus für einen der Kommunikationspartner folgendermaßen dargestellt werden (siehe Abbildung 2.7):

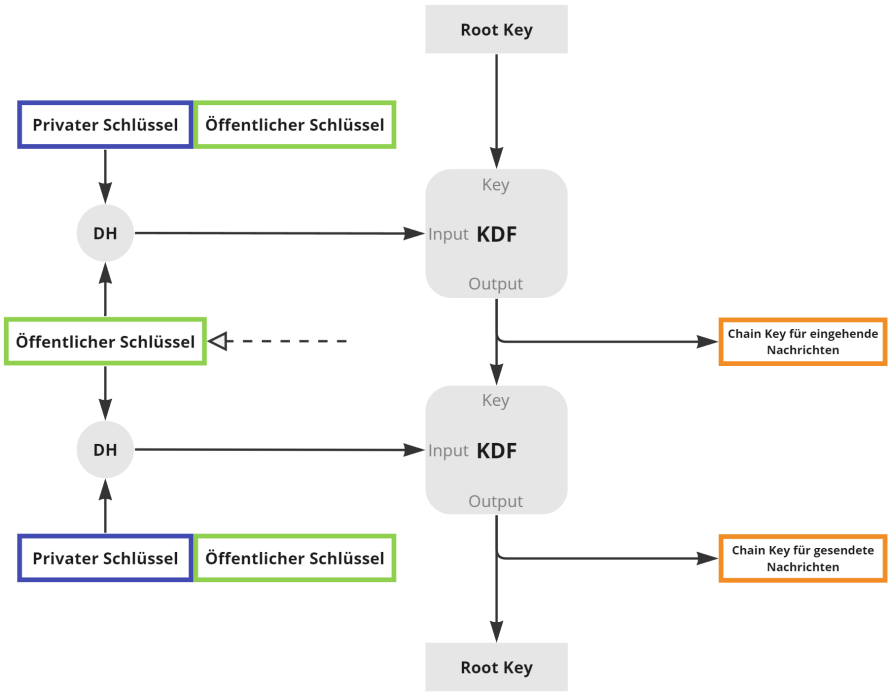

Abbildung 2.7: Vereinfachter Double Ratchet Algorithmus (in Anlehnung an *The Double Ratchet Algorithm* 2016)

Links ist der zweite Ratchet-Schritt zu sehen (asymmetrischer Schlüsselaustausch). In der Abbildung ist dieser mit *DH* gekennzeichnet, was für den Diffie-Hellman-Schlüsselaustausch steht. Auf der rechten Seite befindet sich der erste Ratchet-Schritt. Die Schlüsselableitungsfunktion wird in der Abbildung mit *KDF* bezeichnet. Aus dieser werden die symmetrischen Schlüssel (in der Abbildung als *Chain Key* bezeichnet) abgeleitet, die die jeweiligen Nachrichten ver- und entschlüsseln.

2.4 Blockchain-Technologie

Die bisher beschriebenen Technologien bilden die Grundlage für die sogenannte Blockchain-Technologie. Diese ermöglicht es Daten in einem dezentralen Netzwerk zu speichern und wird unter anderem für Kryptowährungen wie Bitcoin und Ethereum eingesetzt. Christoph Meinel und Tatiana Gayvoronskaya beschreiben die Blockchain-Technologie in ihrem Buch *Blockchain - Hype oder Innovation?* wie folgt:

> *Die Innovation der Blockchain-Technologie ist weder ein neuer Verschlüsselungsalgorithmus noch eine „Alientechnologie“, sondern eine erfolgreiche*

> *Kombination bereits vorhandener technologischen [sic] Ansätze wie Kryptografie, dezentrale Netzwerke und Konsensfindungsmodelle.*
> (Meinel und Gayvoronskaya 2020, S. 17)

In diesem Kapitel wird die Blockchain-Technologie detaillierter betrachtet. Zunächst wird der Begriff Blockchain definiert und anschließend die Funktionsweise erläutert. Daraufhin werden die Konsensmechanismen *Proof-of-Work* und *Proof-of-Stake* vorgestellt. Abschließend wird auf die Sicherheit von Blockchains eingegangen und ausgewählte Angriffe auf Blockchains werden erläutert.

2.4.1 Definition von Blockchain

Der Begriff *Blockchain* setzt sich aus den englischen Wörtern *block* (zu Deutsch: *Block*) und *chain* (zu Deutsch: *Kette*) zusammen. Eine Blockchain ist also eine Kette von Blöcken. Wie in Abbildung 2.8 zu sehen ist, besteht ein Block aus einem *Header* (zu Deutsch: *Kopf*) und einem *Body* (zu Deutsch: *Körper*). Im Kopf des Blocks befinden sich Informationen über den Block und im Körper des Blocks befinden sich die eigentlichen Daten. Diese Daten können beispielsweise Transaktionen sein. Das Format dieser Transaktionen kann je nach Blockchain unterschiedlich sein.

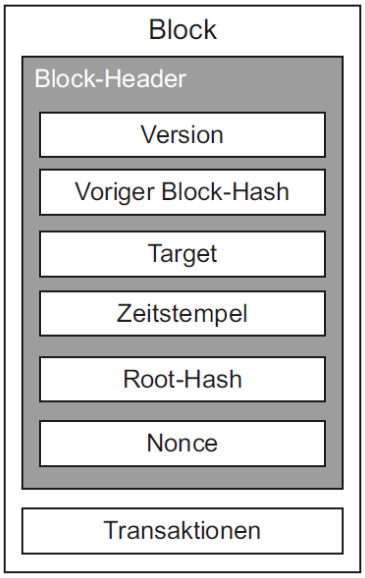

Abbildung 2.8: Aufbau eines Blocks (Fill und Meier 2020, S. 11)

Der Block-Header enthält Informationen, wie zum Beispiel den Hash des vorherigen

Blocks, die Schwierigkeit, den Zeitstempel und die Nonce. Die Schwierigkeit wird durch das *Target*-Feld angegeben und beschreibt, wie schwer es ist, das kryptografische Puzzle zu lösen und damit auch wie schwer es ist, einen neuen Block an die Blockchain anzuhängen. Die Suche nach der Lösung dieses Puzzles wird auch als *Mining* bezeichnet. Der erste, der das Puzzle löst, präsentiert als Beweis dieser Lösung die sogenannte *Nonce*. Die Nonce ist eine zufällige Zahl, die bei der Lösung des Puzzles verwendet wird und aufwändig zu berechnen ist. Nachdem diese neue Version der Blockchain (mit dem neu angehängten Block) an alle anderen Teilnehmer verteilt wurde, kann jeder Teilnehmer überprüfen, ob die Lösung korrekt ist. Durch den Verweis im Block-Header auf den vorherigen Block entsteht die oben erwähnte Kette von Blöcken - die Blockchain (siehe Abbildung 2.9) (Fill und Meier 2020, S. 10-12).

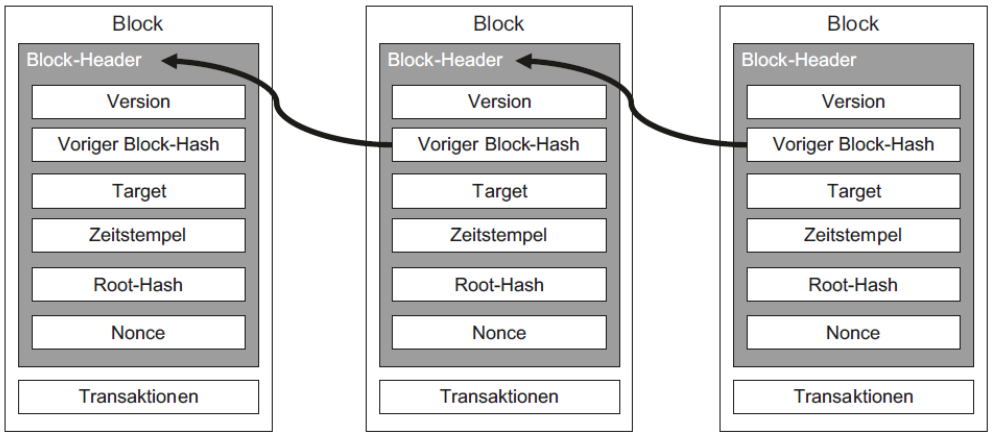

Abbildung 2.9: Kette aus Blöcken (Fill und Meier 2020, S. 12)

2.4.2 Kombination von technologischen Ansätzen

Wie zu Beginn des Abschnitts 2.4 *Blockchain-Technologie* bereits erwähnt, ist die Blockchain das Resultat der Kombination verschiedener Technologien. Diese Technologien werden in diesem Abschnitt erläutert.

Kryptografie

Aus der Kryptografie werden Hash-Funktionen, kryptografische Puzzles, Hash-Bäume und digitale Signaturen verwendet.

Hash-Funktionen gewährleisten die Integrität der Blöcke. Wie dies durch Hashing erreicht wird, wurde in Abschnitt 2.3.2 *Integrität* erläutert.

Kryptografische Puzzles werden dazu verwendet, um die Schwierigkeit beim Mining zu erhöhen. Dabei soll mittels Hash-Funktionen ein bestimmter Ausgabewert gefunden werden. Laut der Definition einer kryptografischen Hash-Funktion (siehe Abschnitt 2.3.2 *Integrität*) ist es nicht möglich, von der Ausgabe auf die Eingabe zu schließen. Daher kann die gesuchte Eingabe nur durch Ausprobieren gefunden werden. Die Schwierigkeit kann durch die Anzahl der Nullen, die am Anfang des Hash-Wertes stehen müssen, angegeben werden. Je mehr Nullen am Anfang des Hash-Wertes stehen müssen, desto schwieriger ist es, die Lösung zu finden, da dadurch der Lösungsraum verkleinert wird (Fill und Meier 2020, S. 6-7; Antonopoulos und Wood 2018, S. 320).

Ein *Merkle-Baum* ist ein binärer Baum, bei dem jeder Knoten den Hash-Wert seiner Kinder enthält. Der nach seinem Erfinder Ralph Merkle benannte Hash-Baum wird in der Blockchain zum Aufbau von Datenstrukturen verwendet. Der Hash-Wert der Wurzel des Baumes wird auch als *Root-Hash* oder *Merkle-Root* bezeichnet (siehe Abbildung 2.8: *Root-Hash*). Wenn sich ein Blatt des Baumes ändert, ändert sich auch der Hash-Wert der Wurzel. Dadurch kann kontrolliert werden, ob sich an einer beliebigen Stelle Daten geändert haben. Wenn sich die Daten geändert haben, ändert sich auch der Root-Hash. Wenn sich die Daten nicht geändert haben, bleibt der Root-Hash gleich. Dadurch kann die Integrität der Daten überprüft werden (Fill und Meier 2020, S. 7-8). In Abbildung 2.10 ist der Aufbau eines Merkle-Baumes zu sehen, der aus vier Transaktionen besteht.

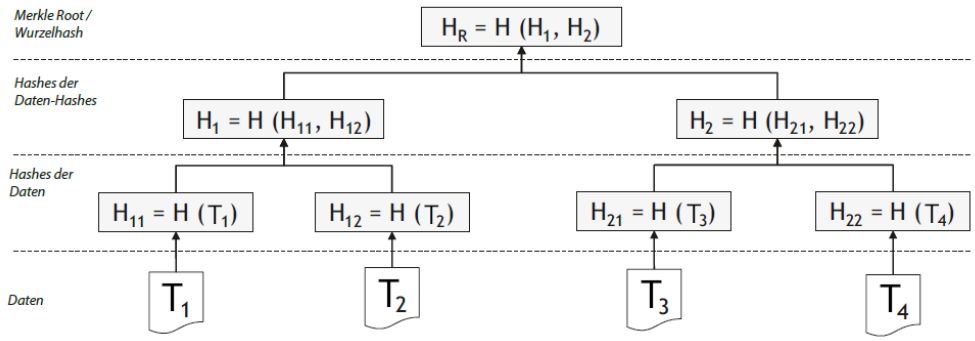

Abbildung 2.10: Aufbau eines Merkle-Baumes (in Anlehnung an Fill und Meier 2020, S. 8)

Von jeder Transaktion T_1, T_2, T_3 und T_4 wird ein Hash-Wert berechnet und in einem Blatt des Baumes gespeichert. Die Hash-Werte der Blätter werden dann paarweise gehasht und in den Knoten darüber gespeichert. Dieser Vorgang wird solange wiederholt, bis nur noch ein Knoten übrig ist. Dieser Knoten enthält den Root-Hash. Sollte sich eine Transaktion ändern, ändert sich auch der Root-Hash. Außerdem kann bewiesen werden,

dass beispielsweise T_2 Teil des Baumes ist, indem mit dem Hash-Wert von T_2 und den Hash-Werten von T_1, T_3 und T_4 versucht wird, den Root-Hash zu berechnen. Wenn der berechnete Root-Hash mit dem tatsächlichen Root-Hash übereinstimmt, ist bewiesen, dass T_2 Teil des Baumes ist (Fill und Meier 2020, S. 9).

Die folgende Abbildung zeigt das Zusammenspiel von Blöcken, Transaktionen und Merkle-Bäumen:

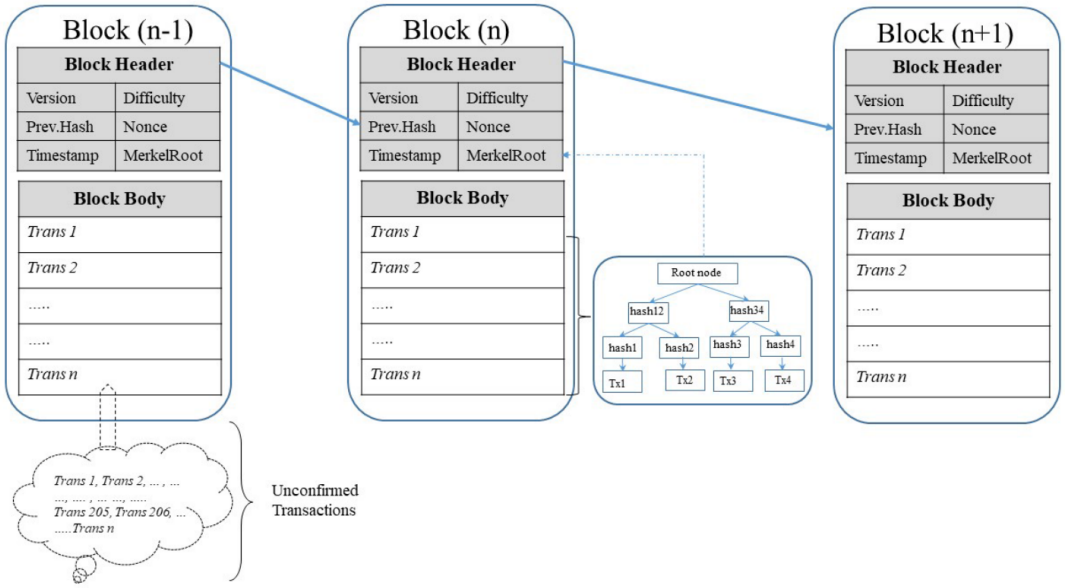

Abbildung 2.11: Verkettete Blöcke mit Transaktionen im Block-Body (Shrestha und Nam 2019, S. 95035)

Eine weitere Technologie aus der Kryptografie, die in der Blockchain verwendet wird, sind digitale Signaturen. Digitale Signaturen werden verwendet, um die Authentizität (siehe 2.3.2 *Integrität*) von Daten zu gewährleisten (Fill und Meier 2020, S. 9-10). Im Falle der Blockchain bedeutet das, dass die Transaktionen, die in einem Block enthalten sind, mit einer digitalen Signatur versehen werden. Dadurch wird den Benutzern der Blockchain garantiert, dass die Transaktionen von dem Besitzer des privaten Schlüssels signiert wurden. Wenn ein Angreifer versucht, die Transaktionen zu manipulieren, wird dies von den anderen Teilnehmern bemerkt, da die Transaktionen nicht mehr mit der digitalen Signatur übereinstimmen (Fill und Meier 2020, S. 10).

Dezentrale Netzwerke

Blockchains machen sich eine weitere Technologie zunutze: dezentrale Netzwerke. Sie basieren auf einem Peer-to-Peer-Netzwerk, bei dem alle Teilnehmer gleichberechtigt sind (siehe 2.2 *Peer-to-Peer-Technologie*). Es gibt keinen zentralen Server, der die Daten verwaltet. Stattdessen werden die Daten auf allen Teilnehmern des Netzwerkes gespeichert. Wenn ein Teilnehmer Daten an das Netzwerk senden möchte, sendet er die Daten an alle anderen Teilnehmer des Netzwerkes. Wenn ein Teilnehmer Daten vom Netzwerk empfangen möchte, empfängt er die Daten von allen anderen Teilnehmern des Netzwerkes. Dadurch ist es nicht möglich, das Netzwerk zu manipulieren, da die Daten auf allen Teilnehmern des Netzwerkes gespeichert sind. Wenn ein Teilnehmer versucht, die Daten zu manipulieren, wird dies von den anderen Teilnehmern bemerkt und die manipulierten Daten werden nicht akzeptiert (Fill und Meier 2020, S. 10/31).

Konsensmechanismen

Durch die Dezentralität der Blockchain ist es möglich, dass die Teilnehmer des Netzwerkes unterschiedliche Daten haben. Es muss also eine Möglichkeit geben, sich auf einen, für alle *richtigen*, gemeinsamen Zustand zu einigen. Dafür werden sogenannte *Konsensmechanismen* verwendet. Dieser Zustand kann beispielsweise die Reihenfolge der getätigten Transaktionen sein. Wenn sich die Teilnehmer nicht auf einen gemeinsamen *Konsens* einigen können, kann das Netzwerk nicht funktionieren. Die Aufgabe der Konsensmechanismen ist es also, einen gemeinsamen Zustand zu finden. Die beiden bekanntesten Konsensmechanismen sind *Proof-of-Work* und *Proof-of-Stake* (Alam 2023, S. 87).

Der *Proof-of-Work* (kurz: *PoW*) ist der Konsensmechanismus, der in der Bitcoin-Blockchain verwendet wird. Bitcoin wurde 2008 von Satoshi Nakamoto, dessen wahre Identität bis heute unbekannt ist, in einem Whitepaper vorgestellt und ist die erste dezentrale Kryptowährung (Nakamoto 2008; S. Zhang und Lee 2019). Der PoW besteht aus dem bereits in Abschnitt 2.4.1 *Definition von Blockchain* beschriebenen kryptografischen Puzzle. Das Lösen des Puzzles durch einen beliebigen Teilnehmer dient als Nachweis für geleistete Rechenarbeit - daher die Bezeichnung *Proof-of-Work* (Brünnler 2018, S. 27). Ein großer Nachteil bei Proof-of-Work ist der hohe Energieverbrauch, der dem hohen Rechenaufwand zum Lösen des Puzzles geschuldet ist (R. Zhang und Chan 2020). Die Website *Digiconomist.net* führt einen Energieindex für Bitcoin, der den Energieverbrauch von Bitcoin in Relation zu verschiedenen Ländern setzt. Aktuell benötigt Bitcoin jährlich rund 144 TWh, was ungefähr dem jährlichen Energieverbrauch von Schweden entspricht (*Bitcoin Energy Consumption Index* 2024).

Der *Proof-of-Stake* (kurz: *PoS*) ist ein alternativer Konsensmechanismus. Im Gegensatz zum PoW wird beim PoS kein kryptografisches Puzzle gelöst, um einen neuen Block anzuhängen. Stattdessen müssen Teilnehmer, die einen neuen Block anhängen möchten, einen Einsatz (engl. *Stake*) in Form von Kryptowährung hinterlegen. Die Wahrscheinlichkeit dafür ist proportional zum hinterlegten Einsatz. Wenn ein Teilnehmer (auch *Validator* genannt) ausgewählt wurde und er einen weiteren Block anhängt, wird dieser von den anderen Validatoren des Netzwerkes überprüft. Wenn der angehängte Block gültig ist, bedeutet das, dass er keine ungültigen Transaktionen enthält und der Teilnehmer seinen Stake wieder erstattet bekommt. Wenn der Block allerdings von den anderen Validatoren als ungültig erkannt wird, verliert der Teilnehmer seinen Stake. Dadurch wird sichergestellt, dass die Teilnehmer des Netzwerkes ehrlich sind und keine ungültigen Blöcke anhängen (Kapengut und Mizrach 2023, S. 96-97; Meinel und Gayvoronskaya 2020, S. 34; Antonopoulos und Wood 2018, S. 320-321).

Die beiden Konsensmechanismen unterscheiden sich grundlegend in ihrer Funktionsweise. PoW, das von Bitcoin genutzt wird, erfordert von Minern, komplexe mathematische Rätsel zu lösen, welche rechenintensiv sind und deshalb viel Energie benötigen. Der Miner, der das Rätsel zuerst löst, kann einen neuen Block hinzufügen und erhält eine Belohnung in Form von Kryptowährung (Fill und Meier 2020). PoW ist bekannt für:

- Energieintensität: PoW erfordert hohe Rechenleistung und verbraucht viel Energie, was Umweltbedenken aufwirft (Kapengut und Mizrach 2023, S. 96-97).
- Sicherheit: Das Netzwerk ist widerstandsfähig gegen Angriffe, da ein Angreifer die Kontrolle über die Mehrheit der Rechenleistung benötigt, um das Netzwerk zu übernehmen, was sehr teuer ist.

Im Gegensatz dazu nutzt PoS den Besitz von Kryptowährung als Sicherheitsfaktor. Validatoren werden auf Basis ihres Einsatzes (Stake) ausgewählt, um Transaktionen zu bestätigen (Meinel und Gayvoronskaya 2020, S. 34; Antonopoulos und Wood 2018, S. 320-321). PoS bietet:

- Energieeffizienz: PoS ist energieeffizienter, da es nicht die immense Rechenleistung von PoW erfordert (Kapengut und Mizrach 2023, S. 96-97).
- Sicherheit durch Einsätze: Die Validatoren haben einen Anreiz, sich ehrlich zu verhalten, da sie ihren Einsatz verlieren könnten, wenn sie betrügen (Meinel und Gayvoronskaya 2020, S. 34; Antonopoulos und Wood 2018, S. 320-321).

Zusammenfassend: PoW ist energieintensiver und bietet Sicherheit durch Rechenleistung, während PoS energieeffizienter ist und Sicherheit durch den Einsatz von Kryptowährung gewährleistet. Beide Mechanismen haben Vor- und Nachteile, und ihre Anwendung hängt von den spezifischen Anforderungen und Zielen des jeweiligen Blockchain-Netzwerks ab.

2.4.3 Sicherheit von Blockchain-Technologie

Die Blockchain gilt als sicher, da sie auf verschiedenen kryptografischen Technologien basiert (siehe Abschnitt 2.4.2 *Kombination von technologischen Ansätzen*). Hinzu kommt die Unveränderlichkeit der Blockchain, was bedeutet, dass einmal geschriebene Transaktionen nach der Bestätigung durch den Konsensmechanismus als unveränderlich gelten und die Blockchain damit eine Fälschungssicherheit bietet (Landerreche und Stevens 2019, S. 1-2; Brünnler 2018, S. 70). Da Blöcke durch kryptografische Hashes verknüpft sind, wäre eine nachträgliche Änderung äußerst rechenintensiv und offensichtlich, da alle nachfolgenden Blöcke ebenfalls verändert werden müssten (Fill und Meier 2020, S. 12). Zusätzlich zu diesen Grundlagen sind Sicherheitsmaßnahmen entscheidend. Anreizstrukturen belohnen Miner/Validatoren für ehrliches Handeln und bestrafen bösartiges Verhalten, was als Abschreckung dient (Antonopoulos und Wood 2018, S. 320-321). Die dezentrale Natur der Blockchain macht sie widerstandsfähig gegen verschiedene Angriffe, da es keinen einzelnen Angriffspunkt, wie beispielsweise einen Server, gibt (Fill und Meier 2020, S. 31).

Letztendlich unterliegt die Sicherheit der Blockchain jedoch keinem absoluten Schutz. Risiken wie 51%-Angriffe oder Schwachstellen in spezifischen Implementierungen können weiterhin Bedrohungen darstellen. Daher sind kontinuierliche Forschung, Protokollaktualisierungen und bewährte Sicherheitspraktiken entscheidend, um Risiken zu minimieren und die Sicherheit der Blockchain weiter zu verbessern (Shrestha und Nam 2019; Perez und Livshits 2019).

2.4.4 Angriffe auf Blockchain

Die Blockchain-Technologie ist nicht immun gegen Angriffe. Es gibt verschiedene Angriffe, die auf Blockchains durchgeführt werden können. Durch die Dezentralität kommen teilweise die gleichen Angriffe wie bei Peer-to-Peer-Netzwerken zum Einsatz. Die bekanntesten Angriffe sind der 51%-Angriff, der Sybil-Angriff und für Ethereum 2.0 die Smart Contract Exploits (Shrestha und Nam 2019, S. 95034; Douceur 2002, S. 251). Diese Angriffe werden in den folgenden Abschnitten erläutert.

51%-Angriff

Ein 51%-Angriff ist ein potenziell bedrohlicher Angriff auf eine Blockchain, bei dem eine einzelne Entität oder eine koordinierte Gruppe die Kontrolle über die Mehrheit der Rechenleistung (bei PoW) oder der Stake (bei PoS) eines Netzwerks erlangt. Dies könnte dazu führen, dass die Angreifer die Blockchain manipulieren, doppelte Ausgaben tätigen (auch als *Double Spending* bekannt) oder Transaktionen zensieren können (Shrestha und Nam 2019, S. 95034; Rosenfeld 2014, S. 2). Dies gestaltet sich als sehr teuer, da die Angreifer die Kontrolle über die Mehrheit der Rechenleistung oder des Stakes erlangen müssen, was sehr viel Energie, Rechenleistung und Geld erfordert. Außerdem ist es sehr unwahrscheinlich, dass ein Angreifer die Kontrolle über die Mehrheit der Rechenleistung oder des Stakes erlangt, da die Blockchain sehr groß ist und es sehr viele Teilnehmer gibt.

Smart Contract Exploits

Smart Contracts sind Programme, die auf der Blockchain ausgeführt werden (siehe 2.4.6 *Smart Contracts*). Sie werden mit Hilfe einer Programmiersprache geschrieben, und wie auch bei jedem anderen Programm können auch Smart Contracts Fehler enthalten. Wenn ein Fehler in einem Smart Contract identifiziert wird, kann dieser Fehler ausgenutzt werden (Perez und Livshits 2019, S. 1-2). Ein Beispiel für einen Smart Contract Exploit ist der *DAO Hack*, der im Juni 2016 stattfand. Der DAO Hack ist ein Angriff auf den Smart Contract *The DAO*, der auf der Ethereum-Blockchain ausgeführt wurde. *DAO* steht für *Decentralized Autonomous Organization* (zu Deutsch: *dezentralisierte autonome Organisation*). Dieser Smart Contract war ein dezentraler Risikokapitalfonds, der es den Teilnehmern ermöglichte, über die Verwendung der Gelder abzustimmen. Die Teilnehmer konnten Ether in den Smart Contract einzahlen und erhielten dafür *DAO-Token*. Diese DAO-Token konnten verwendet werden, um über die Verwendung der Gelder abzustimmen. Wenn ein Teilnehmer seine DAO-Token zurückziehen wollte, konnte er dies tun, indem er den Smart Contract aufrief und seine DAO-Token gegen Ether eintauschte (Pratap 2022).

Experten fanden jedoch eine Schwachstelle im Smart Contract, die es einem Angreifer ermöglichte, mehrmals DAO-Token gegen Ether einzutauschen. Der Angreifer nutzte diese Schwachstelle aus und tauschte seine DAO-Token mehrmals gegen Ether ein. Dadurch erhielt er mehr Ether, als er ursprünglich eingezahlt hatte. Der Angreifer konnte Ether im Wert von 50 Millionen US-Dollar stehlen. Um den Schaden zu begrenzen, wurde ein sogenannter *Hard Fork* durchgeführt. Dabei wurde die Blockchain in zwei Versionen aufgeteilt. In der einen Version wurde der DAO Hack rückgängig gemacht und in der

anderen Version nicht. Die Version, in der der DAO Hack rückgängig gemacht wurde, wurde von der Mehrheit der Teilnehmer verwendet und ist heute als Ethereum bekannt. Die Version, in der der DAO Hack nicht rückgängig gemacht wurde, wird heute als Ethereum Classic bezeichnet. Der DAO Hack ist ein lehrreiches Beispiel dafür, wie ein Fehler in einem Smart Contract ausgenutzt (engl. *exploited*) werden kann, um große Summen an Geld zu stehlen (Price 2016).

Sybil-Angriff

Wie in Abschnitt 2.2.5 *Angriffe auf die Integrität* bereits ausgeführt, ist ein Sybil-Angriff ein Angriff auf ein Peer-to-Peer-Netzwerk, bei dem ein einzelner Angreifer mehrere Identitäten verwendet, um das Netzwerk zu manipulieren. Da Blockchains auf Peer-to-Peer-Netzwerken basieren, sind sie ebenfalls anfällig für diese Art von Angriffen.

2.4.5 Ethereum

Ethereum ist eine der führenden Blockchain-Plattformen und wurde 2013 von Vitalik Buterin entwickelt. Im darauffolgenden Jahr veröffentlichte er seine Idee in einem Whitepaper mit dem Titel *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform*. Darin bezeichnete er die Plattform als *eine dezentrale Struktur mit einer eingebauten Turing-vollständigen Programmiersprache* (Antonopoulos und Wood 2018, S. 3). Im Gegensatz zu Bitcoin, das hauptsächlich als digitale Währung fungiert, ermöglicht Ethereum die Ausführung von Smart Contracts und die Entwicklung von dezentralen Anwendungen (engl. *decentralized Applications*, kurz *DApps*) (Sorge und Krohn-Grimberghe 2013, S. 720; Perez und Livshits 2019, S. 1-2). Bis 2022 wurde als Konsensmechanismus Proof-of-Work verwendet, der jedoch 2022 durch Proof-of-Stake ersetzt wurde. Seitdem existieren zwei Versionen von Ethereum: *Ethereum* und *Ethereum 2.0*. Ethereum verwendet Proof-of-Work, während Ethereum 2.0 Proof-of-Stake verwendet (*Researcher FAQs* o. D.).

Ether (ETH) ist die native Kryptowährung von Ethereum und wird für Transaktionen innerhalb des Netzwerks verwendet. Es dient auch als Anreiz für diejenigen, die an der Sicherung des Netzwerks durch Mining (bei PoW) oder Validierung (bei PoS) von Transaktionen (siehe 2.4.2 *Konsensmechanismen*) beteiligt sind (Antonopoulos und Wood 2018, S. 320-321). Die Flexibilität von Ethereum und seine Fähigkeit, innovative Lösungen zu unterstützen, haben es zu einer der wichtigsten Plattformen in der Blockchain-Welt gemacht.

2.4.6 Smart Contracts

Ein Smart Contract (zu Deutsch: *intelligenter Vertrag*) ist im Grunde genommen ein selbstausführender Vertrag, der automatisch Aktionen auslöst, wenn bestimmte Bedingungen erfüllt sind (Perez und Livshits 2019, S. 1-2). Die Bezeichnung *Smart Contract* ist eigentlich eine Fehlbezeichnung, da es sich weder um einen Vertrag im rechtlichen Sinne, noch um einen *intelligenten* Vertrag handelt, doch der Begriff hat sich in der Blockchain-Community etabliert und wird deshalb weiterhin verwendet. Ein Lesezugriff auf einen Smart Contract ist kostenlos, ein Schreibzugriff hingegen kostet Geld, da die Transaktion in der Blockchain gespeichert werden muss. Dieses Geld wird als *Gas* bezeichnet und ist eine Art Gebühr, die gezahlt werden muss, um die Rechenleistung des Netzwerks zu nutzen (Antonopoulos und Wood 2018, S. 127). Um Gas zu erhalten, muss der Nutzer Ether eintauschen, die Währung der Ethereum-Blockchain. Für das Protokoll dieser Arbeit wurden sowohl Lese- als auch Schreibzugriffe auf Smart Contracts implementiert (siehe Kapitel 4 *Architektur des Protokolls*).

Die Plattform verwendet die objektorientierte Programmiersprache *Solidity*, die speziell für Smart Contracts entwickelt wurde und stark an JavaScript angelehnt ist. Entwickler können mit Hilfe von Solidity Smart Contracts erstellen, die dann in der Ethereum-Blockchain ausgeführt werden und von jedem Teilnehmer des Netzwerks aufgerufen werden können (Antonopoulos und Wood 2018, S. 127-133).

Kapitel 3

Anforderungsanalyse

Die Anforderungsanalyse dient dazu, die Grundlage für den erfolgreichen Verlauf eines Softwareprojekts zu schaffen, indem sie sicherstellt, dass die Ziele und Anforderungen des Projekts klar definiert und verstanden werden (Zakharyan o. D.). In dieser Arbeit wird ein Prototyp für ein Peer-to-Peer-Instant-Messaging-Protokoll entwickelt. Das Ziel dieser Arbeit ist es, die Machbarkeit eines solchen Protokolls aufzuzeigen, weshalb die Anforderungen an dieses Protokoll klar definiert werden müssen. Dazu werden in diesem Kapitel die funktionalen und nicht-funktionalen Anforderungen an das Protokoll beschrieben.

3.1 Funktionale Anforderungen

Funktionale Anforderungen beziehen sich auf die spezifischen Funktionen und Aufgaben, die eine Software oder ein System erfüllen muss, um die Bedürfnisse und Erwartungen der Benutzer zu erfüllen. Sie beschreiben, was das System tun soll, welche Aktionen es ausführen muss und welche Ergebnisse es liefern sollte. Mitunter werden sie auch dazu verwendet, um festzuhalten, was das System nicht können soll. All diese Anforderungen sind entscheidend, um sicherzustellen, dass die entwickelte Software oder wie in diesem Fall, das entwickelte Protokoll die erwarteten Funktionen erbringt. Sie dienen als Grundlage für das Design, die Entwicklung, die Validierung und die Verifizierung von Software-Systemen und sind ein wichtiger Bestandteil des Anforderungsmanagements im Software-Engineering-Prozess (Sommerville 2018, S. 124-126).

Um die Anforderungen an das zu entwickelnde Protokoll zu definieren, wird es in die folgenden Funktionen unterteilt:

- Peer-Discovery und Routing

- Verbindungsmanagement
- Nachrichtenformatierung
- Sicherheit und Verschlüsselung
- Plattformunabhängigkeit
- Fehlerbehandlung und Wiederholungsmechanismen

Die folgenden Abschnitte beschreiben die funktionalen Anforderungen an das Protokoll.

3.1.1 Peer-Discovery und Routing

Das Protokoll muss es den Benutzern ermöglichen, sich gegenseitig zu finden, um miteinander kommunizieren zu können. Als Grundlage hierfür soll das Internet dienen. Das bedeutet, dass das Protokoll auf IP-Adressen basieren muss. Dazu muss es eine Möglichkeit geben, die IP-Adresse eines Benutzers zu ermitteln, wenn der Benutzername des Ziels bekannt ist. Wenn die IP-Adresse eines Benutzers bekannt ist, muss das Protokoll in der Lage sein, eine Verbindung zu diesem Benutzer herzustellen. Dies ist notwendig, um die Dezentralität des Protokolls zu gewährleisten. Sollte ein Art von zentralem Server benötigt werden, muss die Verschlüsselung der Nachrichten so implementiert werden, dass dem Server nicht vertraut werden muss und dieser damit nicht in der Lage ist, die Nachrichten zu entschlüsseln und zu lesen.

3.1.2 Verbindungsmanagement

Das Verbindungsmanagement in einem Peer-to-Peer Instant-Messaging-Protokoll umfasst verschiedene Aspekte, die für eine zuverlässige, stabile und sichere Kommunikation zwischen den Peers essentiell sind.

Zunächst spielt der Verbindungsaufbau eine wichtige Rolle. Dieser Mechanismus ermöglicht es den Peers, miteinander in Verbindung zu treten. Durch die Verwendung von IP-Adressen und Ports oder anderen Identifikationsmechanismen wird sichergestellt, dass die Kommunikation initiiert werden kann. Dieser Prozess sollte sicher und authentifiziert ablaufen, um die Integrität des Netzwerks zu gewährleisten. Die Überwachung der Verbindungsstabilität ist ein weiterer wichtiger Aspekt des Verbindungsmanagements. Durch regelmäßige Überprüfungen sollte sichergestellt werden, dass die Verbindung zwischen den Peers aktiv bleibt und eventuelle Probleme frühzeitig erkannt werden können. Ein weiterer Schlüsselaspekt ist die Verbindungsbeendigung. Ein ordnungsgemäßer Mechanismus

zur Beendigung von Verbindungen ist wichtig, um Ressourcen freizugeben und mögliche Sicherheitsrisiken zu minimieren. Die Verbindungsbeendigung kann durch Benutzeraktionen wie das Abmelden ausgelöst werden oder aufgrund von Fehlern im Netzwerk auftreten. Im Falle vorübergehender Unterbrechungen, beispielsweise durch Netzwerkausfälle, sollte das Verbindungsmanagement Mechanismen zur automatischen Wiederherstellung von Verbindungen bereitstellen. Die Verbindungsauthentifizierung hingegen stellt sicher, dass die Kommunikation nur zwischen vertrauenswürdigen Parteien stattfindet. Dieser Sicherheitsaspekt ist entscheidend, um unautorisierte Zugriffe zu verhindern und die Vertraulichkeit der übertragenen Daten zu gewährleisten.

3.1.3 Nachrichtenformatierung

Die Nachrichtenformatierung ist ein wichtiger Aspekt eines Instant-Messaging-Protokolls. Sie definiert, wie die Nachrichten strukturiert sind und welche Informationen sie enthalten. Die Nachrichtenformatierung ist entscheidend für die Funktionalität des Protokolls, da sie die Grundlage für die Kommunikation zwischen den Peers bildet. Die Nachrichtenformatierung muss so gestaltet sein, dass sie die folgenden Anforderungen erfüllt:

- Die Nachrichten müssen in einem standardisierten Format vorliegen, um die Interoperabilität zwischen den verschiedenen Implementierungen des Protokolls zu gewährleisten.
- Die Nachrichten müssen die erforderlichen Informationen enthalten, um die Kommunikation zwischen den Peers zu ermöglichen.
- Die Nachrichten müssen so strukturiert sein, dass sie von den Peers verarbeitet werden können.

3.1.4 Sicherheit und Verschlüsselung

Es müssen Mechanismen zur Verschlüsselung der Kommunikation zwischen den Peers implementiert werden, um die Vertraulichkeit der Nachrichten zu gewährleisten. Eine Ende-zu-Ende-Verschlüsselung ist erforderlich, um sicherzustellen, dass die Nachrichten nur vom Absender und Empfänger gelesen werden können. Das Netzwerk muss so gestaltet sein, dass es vor den gängigen Angriffen eines Peer-to-Peer-Netzwerks geschützt ist. Dazu gehören unter anderem Sybil-Angriffe, Eclipse-Angriffe und Routing-Angriffe.

3.1.5 Plattformunabhängigkeit

Das Protokoll sollte auf verschiedenen Betriebssystemen und Gerätetypen nahtlos funktionieren. Um Plattformunabhängigkeit zu gewährleisten, sollte das Protokoll auf offenen, standardisierten Technologien basieren. Hierzu gehören beispielsweise Netzwerkprotokolle wie TCP, UDP oder IP. Die Verwendung plattformübergreifender Standards stellt sicher, dass die Kernfunktionalitäten des Protokolls von den meisten Betriebssystemen unterstützt werden. Die Implementierung der Protokollspezifikationen sollte in verschiedenen Programmiersprachen möglich sein. Dies ermöglicht es, das Protokoll in verschiedenen Anwendungen zu verwenden. Diese Unabhängigkeit ist ein wichtiger Aspekt, um die Verbreitung des Protokolls zu fördern und die Interoperabilität zwischen verschiedenen Anwendungen zu gewährleisten.

3.1.6 Fehlerbehandlung und Wiederholungsmechanismen

Die Fehlerbehandlung und die Implementierung von Wiederholungsmechanismen stellen entscheidende Komponenten im Verlauf einer Peer-to-Peer Instant-Messaging-Kommunikation dar. Ein zentraler Aspekt der Fehlerbehandlung ist die Erkennung von Übertragungsfehlern. Das Protokoll sollte in der Lage sein, Fehlerzustände während des Nachrichtenaustauschs zu identifizieren. Dies können beispielsweise fehlerhafte Pakete, verlorene Verbindungen oder andere unvorhergesehene Probleme sein. Die Fehlererkennung ermöglicht es, schnell auf Probleme zu reagieren und entsprechende Maßnahmen einzuleiten. Die Wiederholungsmechanismen sind eng mit der Fehlerbehandlung verbunden und dienen dazu, sicherzustellen, dass fehlgeschlagene Übertragungen erneut versucht werden. Dies könnte durch automatisches erneutes Senden von Nachrichten oder das Auslösen spezifischer Protokollmechanismen geschehen. Die Wiederholungsmechanismen sind darauf ausgerichtet, die Zustellung von Nachrichten trotz vorübergehender Probleme im Netzwerk zu gewährleisten.

3.2 Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen sind Anforderungen, die sich nicht auf eine spezifische Funktionalität einer Software-Anwendung beziehen, sondern auf die Gesamtstruktur und damit auf Qualitätsmerkmale und Aspekte, wie beispielsweise die Leistung, Zuverlässigkeit, Reaktionszeit und Benutzerfreundlichkeit der Software. Diese Anforderungen beschreiben, *wie* die Software funktionieren sollte, anstatt *was* sie tun sollte. Nichtfunktionale Anforderungen sind genauso wichtig wie funktionale Anforderungen, da sie

einen erheblichen Einfluss auf die Gesamtleistung und die Benutzerzufriedenheit haben können (Sommerville 2018, S. 126-130).

Die folgenden nicht-funktionale Anforderungen sollte das Protokoll bieten:

- Performanz
- Sicherheit
- Zuverlässigkeit
- Kompatibilität
- Skalierbarkeit

Die folgenden Abschnitte beschreiben die nicht-funktionalen Anforderungen an das Protokoll.

3.2.1 Performanz

Die Leistungsanforderungen an ein Peer-to-Peer Instant-Messaging-Protokoll beziehen sich auf die Effizienz und Geschwindigkeit der Nachrichtenübertragung, um sicherzustellen, dass Benutzer eine reaktionsschnelle und nahtlose Kommunikation erfahren. Die Leistungsfähigkeit des Protokolls beeinflusst direkt die Benutzerzufriedenheit und die Gesamterfahrung der Instant-Messaging-Anwendung, die das Protokoll verwendet.

3.2.2 Sicherheit

Das entwickelte Protokoll möglichst sicher sein, um die Vertraulichkeit, Integrität und Authentizität der übertragenen Nachrichten und der Benutzerdaten gewährleisten zu können. Ein sicherheitsorientiertes Protokoll minimiert das Risiko von unbefugtem Zugriff, Datenmanipulation oder anderen Bedrohungen, die die Vertraulichkeit und Integrität der Kommunikation gefährden könnten.

3.2.3 Zuverlässigkeit

Die Zuverlässigkeit des zu entwickelnden Instant-Messaging-Protokolls bezieht sich darauf, dass das System kontinuierlich und konsistent arbeitet, selbst unter verschiedenen Lastszenarien. Eine zuverlässige Kommunikation ist essentiell, um sicherzustellen, dass Nachrichten korrekt zugestellt werden und Benutzer jederzeit auf den Instant-Messaging-Dienst zugreifen können.

3.2.4 Kompatibilität

Die Kompatibilität eines Peer-to-Peer-Instant-Messaging-Protokolls bezieht sich darauf, wie gut es mit verschiedenen Plattformen, Betriebssystemen und Anwendungen zusammenarbeitet. Ein kompatibles Protokoll ermöglicht die nahtlose Kommunikation zwischen Benutzern, unabhängig von den verwendeten Geräten oder Softwareanwendungen. Dies schließt die Unterstützung unterschiedlicher Betriebssysteme, Interoperabilität mit anderen Messaging-Diensten sowie die Integration in verschiedene Anwendungen ein.

3.2.5 Skalierbarkeit

Die Skalierbarkeit eines Peer-to-Peer Instant-Messaging-Protokolls ist entscheidend, um sicherzustellen, dass die Kommunikationseffizienz und -qualität auch bei steigender Benutzerzahl erhalten bleibt. Skalierbarkeit bezieht sich auf die Fähigkeit des Protokolls, mit einem zunehmenden Benutzerwachstum umzugehen, ohne dabei signifikante Einbußen in Bezug auf Leistung und Reaktionsfähigkeit zu erleiden.

Kapitel 4

Architektur des Protokolls

In diesem Kapitel wird die Architektur des Protokolls entwickelt. Dazu wird zunächst eine geeignete Peer-to-Peer-Technologie ausgewählt. Anschließend wird die darauf aufbauende Peer-Discovery und das Routing betrachtet. Danach wird das Verbindungsmanagement, welches den Verbindungsaufbau, die Nachrichtenübertragung und den Verbindungsabbau beinhaltet, definiert. Darauf folgt ein Vorschlag einer Definition für eine Ende-zu-Ende-Verschlüsselung, welche die Vertraulichkeit der Nachrichten gewährleistet. Abschließend werden verschiedene Möglichkeiten zur Integration der Blockchain in das Protokoll besprochen und geeignete davon ausgewählt.

4.1 Auswahl der Peer-to-Peer-Technologie

Das Tox-Protokoll, das auch auf Peer-to-Peer-Technologie aufbaut (siehe 2.1.3 *Tox*), verwendet zur Auffindung der Teilnehmer verteilte Hashtabellen (zetok o. D.). Da dies eine effiziente Möglichkeit ist, um Teilnehmer zu finden, wird diese Technologie auch für das Protokoll dieser Arbeit verwendet. Verteilte Hashtabellen haben allerdings den Nachteil, das der Verbindungsaufbau im modernen Internet durch NATs erschwert wird. Deshalb wird für das entwickelte Protokoll ein mehrstufiger Ansatz verwendet. In den folgenden Abschnitten wird der Ansatz erläutert und die verwendeten Technologien beschrieben.

Für den effizienten Aufbau einer Direktverbindung zwischen zwei Teilnehmern kamen Chord und Kademlia in die engere Auswahl, welche beide lange Gegenstand intensiver Forschung waren, sowohl in der Industrie als auch in der akademischen Welt (Medrano-Chávez, Perez-Cortès und Lopez-Guerrero 2015, S. 808). Das Chord-Protokoll und das Kademlia-Protokoll sind zwei grundlegend verschiedene Ansätze zur Organisation von Peer-to-Peer-Netzwerken. Beide Protokolle sind strukturiert und bieten eine effiziente

Ressourcenlokalisierung, aber sie unterscheiden sich in ihrer Routing-Struktur und der Art und Weise, wie sie die Knoteninformationen verwalten.

Chord

Chord basiert auf einer Ringstruktur (siehe Abbildung 4.1), bei der die Knoten in einem Ring angeordnet sind und jeder Knoten für einen bestimmten Schlüsselbereich verantwortlich ist. Die Verbindungen zwischen den Knoten sind durch ihren Platz im Ring definiert, wobei jeder Knoten eine Verbindung zu seinem nächsten Nachbarn im Uhrzeigersinn hat. Ein Knoten besitzt zwei Informationsmengen: eine *Successor-Liste* und eine *Finger-Tabelle*. Die Successor-Liste enthält die Knoten, die direkt nach dem Knoten im Uhrzeigersinn im Ring kommen. Die Anzahl der dort enthaltenen Knoten hängt davon ab, wie viele Knoten im Netzwerk insgesamt vorhanden sind. Die Finger-Tabelle enthält die Knoten, die für die Schlüsselbereiche verantwortlich sind, die durch eine Berechnung auf der ID des Knotens basieren (Medrano-Chávez, Perez-Cortès und Lopez-Guerrero 2015, S. 810-811).

Wenn das Chord-Netzwerk eine Suchanfrage erhält, gibt es zwei Strategien, um die Anfrage zu bearbeiten. In der ersten Strategie wird die Anfrage sequentiell von Knoten zu Knoten weitergeleitet, bis der Knoten gefunden wird, der für den Schlüsselbereich verantwortlich ist, in dem sich der gesuchte Schlüssel befindet. Für diese Suchstrategie ergibt sich daher eine Komplexität von $\mathcal{O}(n)$, wobei n die Anzahl der Knoten im Netzwerk ist. $\mathcal{O}(n)$ beschreibt eine lineare Komplexität. Das bedeutet, dass die Anzahl der Weiterleitungen von der Anzahl der Knoten im Netzwerk abhängt (Medrano-Chávez, Perez-Cortès und Lopez-Guerrero 2015, S. 810-811).

Die zweite Strategie verwendet die Finger-Tabelle, um die Anzahl der Knoten zu reduzieren, die die Anfrage weiterleiten. Diese Strategie hat eine Komplexität von $\mathcal{O}(\log n)$, wobei n die Anzahl der Knoten im Netzwerk ist. Hier beschreibt $\mathcal{O}(\log n)$ eine logarithmische Komplexität. Das bedeutet, dass die Anzahl der Weiterleitungen von der Anzahl der Knoten im Netzwerk abhängt, aber nicht linear, sondern logarithmisch. Dies ist eine effizientere Strategie als die sequentielle Weiterleitung (Medrano-Chávez, Perez-Cortès und Lopez-Guerrero 2015, S. 810-811).

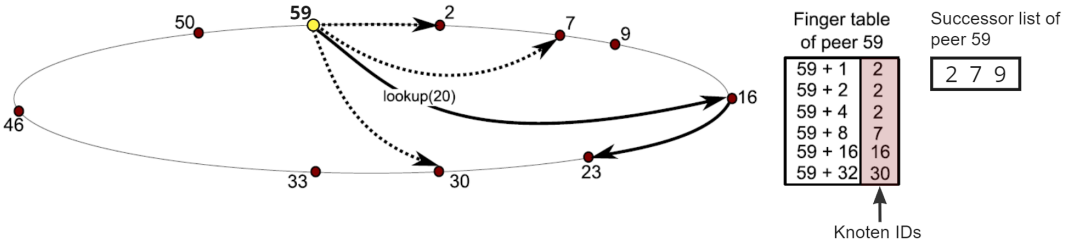

Abbildung 4.1: Visualisierung einer Suche in der Ringstruktur von Chord (in Anlehnung an Medrano-Chávez, Perez-Cortès und Lopez-Guerrero 2015, S. 811)

In Abbildung 4.1 ist zu erkennen, dass Knoten 59 eine Suchanfrage für den Knoten mit der ID 20 beginnt. Unter Verwendung der Finger-Tabelle von Knoten 59 wird die Anfrage an den Knoten gesendet, der am nächsten an Knoten 20 liegt. In diesem Fall ist dies Knoten 16. Knoten 16 wiederum leitet die Anfrage an den Knoten weiter, der ebenfalls am nächsten an Knoten 20 liegt, was Knoten 23 ist. Knoten 23 ist für den Schlüsselbereich verantwortlich, in dem sich der gesuchte Schlüssel befindet, und sendet daher die Antwort an Knoten 59 zurück. Durch die Verwendung dieser Strategie wurde Knoten 20 in nur zwei Schritten gefunden, anstatt in fünft Schritten, wenn die Anfrage sequentiell weitergeleitet worden wäre.

Kademlia

Im Gegensatz zu Chord verwendet Kademlia eine sogenannte k-Bucket-Struktur, die in Abbildung 4.2 zu sehen ist, um eine effiziente Verwaltung von Knoteninformationen zu ermöglichen. Die k-Buckets enthalten eine Liste von Knoten für verschiedene Schlüsselbereiche basierend auf ihrer Nähe, die durch XOR-Distanzen der IDs berechnet wird. Die Verbindungen zwischen den Knoten sind asymmetrisch (siehe 2.2.4 *Kademlia vs. Chord vs. Pastry*), und jeder Knoten speichert Informationen über andere Knoten in seinen k-Buckets. Bei der Suche nach einem bestimmten Schlüssel erfolgt das Routing durch die XOR-Entfernung, wodurch die nächsten Knoten für diesen Schlüssel gefunden werden.

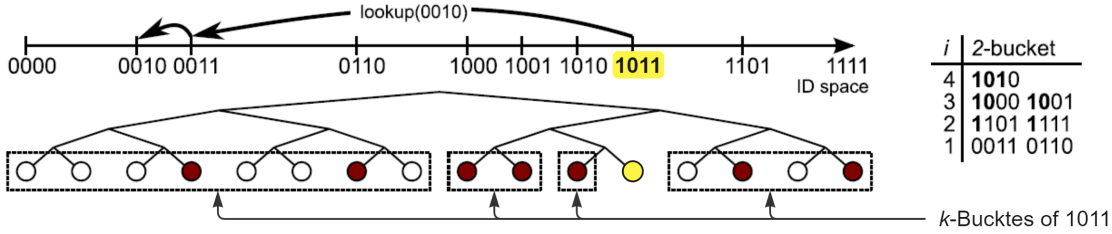

Abbildung 4.2: Visualisierung der Baumstruktur von Kademlia, in Anlehnung an Medrano-Chávez, Perez-Cortès und Lopez-Guerrero 2015, S. 812

In Abbildung 4.2 ist zu sehen, wie der Knoten 1011 ein Suche nach Knoten 0010 startet. Der Knoten 1011 sucht in seinen vier Buckets nach dem nächsten Knoten, der am Schlüssel 0010 liegt. Im ersten Bucket sind Knoten mit dem Präfix $0xxx$ enthalten. In Bucket zwei sind Knoten mit dem Präfix $11xx$, in Bucket drei Knoten mit dem Präfix $100x$ und in Bucket vier Knoten mit dem Präfix $101x$. Da der Schlüssel 0010 mit dem Präfix 00 beginnt, wird der nächste Knoten für diesen Schlüssel im ersten Bucket gesucht. Knoten 0011 wird als nächster Knoten für den Schlüssel 0010 gefunden. Knoten 1011 sendet die Anfrage also an Knoten 0011, der wiederum den nächsten Knoten für den Schlüssel 0010 sucht. Da dieser Knoten den Schlüssel 0010 in einem seiner Buckets besitzt, sendet er die Antwort auf die Suchanfrage an Knoten 1011 zurück. Es werden zwei Weiterleitungen durchgeführt, um den Zielknoten zu finden, woraus sich eine Komplexität von $\mathcal{O}(\log n)$ ergibt, wobei n die Anzahl der Knoten im Netzwerk ist. Dies stellt, wie bereits erwähnt, eine logarithmische Komplexität dar, was bedeutet, dass die Anzahl der Weiterleitungen von der Anzahl der Knoten im Netzwerk abhängt, aber nicht linear, sondern logarithmisch. Dies wiederum bedeutet, dass die Anzahl der Weiterleitungen bei einer großen Anzahl von Knoten im Netzwerk nicht stark ansteigt (Medrano-Chávez, Perez-Cortès und Lopez-Guerrero 2015, S. 812).

Funktionen, wie diese Suche, werden durch die vier Nachrichten *FIND_ NODE*, *FIND_ VALUE*, *PING* und *STORE* des Kademlia-Protokolls ermöglicht. Sie haben die folgende Verwendung (Maymounkov und Mazières 2002, S. 3):

- *FIND_ NODE*: Diese Nachricht wird verwendet, um den nächsten Knoten für einen bestimmten Schlüssel zu finden. Sie wird von einem Knoten an einen anderen Knoten gesendet, der den Schlüssel in seinem k-Bucket hat. Der Knoten, der die Nachricht erhält, antwortet mit einer Liste von Knoten, die den Schlüssel in ihrem k-Bucket haben (Maymounkov und Mazières 2002, S. 3).

- *FIND_ VALUE*: Diese Nachricht wird verwendet, um den Wert für einen bestimmten Schlüssel zu finden. Sie wird von einem Knoten an einen anderen Knoten gesendet, der den Schlüssel in seinem k-Bucket hat. Der Knoten, der die Nachricht erhält, antwortet mit dem Wert, der dem Schlüssel zugeordnet ist, oder mit einer Liste von Knoten, die den Schlüssel in ihrem k-Bucket haben (Maymounkov und Mazières 2002, S. 3).

- *PING*: Diese Nachricht wird verwendet, um die Erreichbarkeit eines Knotens zu

überprüfen. Sie wird von einem Knoten an einen anderen Knoten gesendet, um zu überprüfen, ob der Knoten noch erreichbar ist (Maymounkov und Mazières 2002, S. 2-3).

- *STORE*: Diese Nachricht wird verwendet, um einen Schlüssel-Wert-Paar in einem k-Bucket zu speichern. Sie wird von einem Knoten an einen anderen Knoten gesendet, um den Schlüssel-Wert-Paar in seinem k-Bucket zu speichern. Der Knoten, der die Nachricht erhält, speichert den Schlüssel-Wert-Paar in seinem k-Bucket (Maymounkov und Mazières 2002, S. 3).

Da bei einem Instant-Messaging-Protokoll häufig Teilnehmer das Netzwerk verlassen und neue Teilnehmer dem Netzwerk beitreten, ist es wichtig, dass das Protokoll mit hoher Fluktuation umgehen kann. Diese Fluktuation von Nodes wird als Churn bezeichnet. In einer Studie von Medrano-Chávez et al. (Medrano-Chávez, Perez-Cortès und Lopez-Guerrero 2015), welche im hybriden Journal *Peer-to-Peer Networking and Applications* veröffentlicht wurde, wurde die Leistung von Chord und Kademlia in Bezug auf Netzwerkfluktuation untersucht.

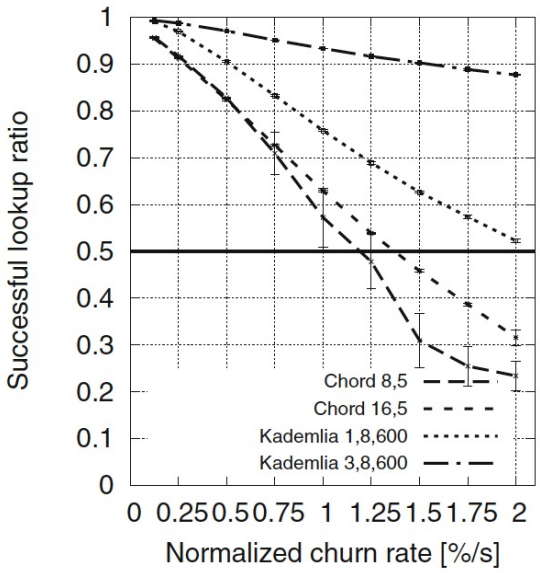

Abbildung 4.3: Vergleich der Leistung von Chord und Kademlia bei hoher Fluktuation (Medrano-Chávez, Perez-Cortès und Lopez-Guerrero 2015, S. 818)

In Abbildung 4.3 ist ein Graph zu sehen, auf dem vier Linien zu sehen sind. Auf der x-Achse ist die ist die Churn-Rate in Fluktuationen pro Sekunde aufgetragen. Auf der y-Achse ist das Verhältnis der erfolgreichen Suchanfragen zur Anzahl der gesendeten

Suchanfragen aufgetragen. Die beiden oberen Linien zeigen die Ergebnisse für Kademlia, die beiden unteren Linien die Ergebnisse für Chord. Die Ergebnisse aus Abbildung 4.3 zeigen, dass Kademlia bei hoher Fluktuation besser abschneidet als Chord. Aus diesem Grund wird Kademlia in diesem Protokoll als Grundlage für das Auffinden von Teilnehmern verwendet.

Sollte der Aufbau einer Direktverbindung mittels Kademlia nicht möglich sein, wird das Interactive Connectivity Establishment (ICE) Protokoll verwendet, um eine Verbindung zwischen zwei Teilnehmern herzustellen, da es mehrere Techniken kombiniert, um eine Verbindung zwischen zwei Endpunkten herzustellen, die sich hinter NATs befinden. Die detaillierte Beschreibung von ICE folgt in Kapitel 4.3 *Verbindungsmanagement*.

4.2 Peer-Discovery und Routing

Da es in Peer-to-Peer-Netzwerken keinen zentralen Server gibt, der unter anderem zur Auffindung und Identifikation anderer Teilnehmer verwendet werden kann, müssen die Teilnehmer auf andere Weise identifiziert werden. Da sich im vorangegangenen Abschnitt für die Implementierung von Kademlia entschieden wurde, wird in diesem Abschnitt beschrieben, wie die Teilnehmer identifiziert werden. Um einen Teilnehmer zu identifizieren, wird eine ID benötigt. Hierfür wird der Benutzername des Teilnehmers verwendet, welchen er bei der Registrierung frei wählen kann.

Aus der Spezifikation von Kademlia geht hervor, dass die ID einer Node, welche auch als *Kademlia-ID* bezeichnet wird, aus einer zufälligen Kennung bestehen soll (Maymounkov und Mazières 2002, S. 2). Für das hier entwickelte Protokoll wird, wie bereits oben festgelegt, der Benutzername als ID verwendet. Das führt dazu, dass der Benutzername eindeutig sein muss, da sonst mehrere Teilnehmer die gleiche ID haben könnten. Um den Benutzer später identifizieren zu können, wird bei der Registrierung ein statisches Schlüsselpaar generiert. Der öffentliche Schlüssel wird zusammen mit dem Benutzernamen in der Blockchain gespeichert (siehe 4.6.4 *Registrierung*). Der private Schlüssel wird lokal auf dem Gerät des Teilnehmers gespeichert.

Außerdem wird eine ID mit einer Länge von 160 Bit erwartet. Daher muss der Benutzername auf diese Länge gebracht werden. Dafür wird eine Zeichenbegrenzung von 20 Zeichen festgelegt, was bei einer UTF-8 Codierung einer Bitlänge von 160 Bit entspricht (Yergeau 2003). Falls der Benutzername kürzer als 20 Zeichen ist, wird er mit Nullen aufgefüllt. Durch die Beschränkung der ID auf 20 Zeichen wird die Anzahl der Teilnehmer zwar begrenzt, jedoch ist die Anzahl der möglichen Teilnehmer immer noch sehr groß. Es können in der Theorie 2^{160} Teilnehmer am Netzwerk teilnehmen, wobei nicht

jedes Zeichen in UTF-8 sinnvoll für die Verwendung in einem Benutzername ist. Somit ist die Anzahl der möglichen Teilnehmer in der Praxis geringer, aber immer noch groß genug, um einem Instant-Messaging-Protokoll zu genügen. In der Praxis sollte diese Zahl nicht erreicht werden, wodurch sich die Entscheidung, die Länge des Benutzernamens zu beschränken, nicht negativ auf die Funktionalität des Protokolls auswirken sollte. In der Node werden dann Schlüssel-Wert-Paare gespeichert, wobei der Schlüssel die ID des Teilnehmers ist und der Wert die IP-Adresse und der Port. Es wird festgelegt, dass immer Port 49152 verwendet wird, da sich dieser Port im Bereich der dynamischen Ports befindet und somit nicht für andere Anwendungen reserviert ist (Cotton u. a. 2011, S. 20).

Wenn ein Teilnehmer eine Nachricht an einen anderen Teilnehmer senden möchte, muss er dessen IP-Adresse kennen. Um an diese Information zu gelangen, muss zuerst die ID des Teilnehmers in der Blockchain gesucht werden und anschließend eine Peer-Discovery durchgeführt werden, um die IP-Adresse des Teilnehmers zu erhalten. Eine Peer-Discovery läuft wie folgt ab:

1. Alice sendet eine *FIND_NODE*-Nachricht an den Knoten in ihrer Routing-Tabelle, der der Kademlia-ID von Bob am nächsten ist.

2. Der Knoten sucht in seiner Routing-Tabelle nach der Kademlia-ID von Bob. Falls er den Knoten findet, antwortet er mit den gefundenen Informationen in Form eines Tripletts aus Kademlia-ID, IP-Adresse und Port. Falls dieser Knoten den gesuchten Knoten nicht findet, antwortet er mit mehreren Triplets, die dem Knoten mit der Kademlia-ID von Bob am nächsten sind.

3. Alice erhält die Antwort und sucht in der Antwort nach der Kademlia-ID von Bob. Falls sie gefunden wird, speichert Alice die IP-Adresse und den Port von Bob in ihrer Routing-Tabelle ab. Falls sie nicht gefunden wird, wiederholt Alice Schritt 2 solange, bis sie keine Antworten mehr erhält, die IDs beinhalten, die der Kademlia-ID von Bob näher sind als die ID des Knotens, der die *FIND_NODE*-Nachricht erhalten hat.

Um das Netzwerk immer aktuell zu halten, wird in regelmäßigen Abständen ein *PING* an alle Teilnehmer gesendet, die in den Routing-Tabellen der Nodes gespeichert sind. Falls ein Teilnehmer nicht antwortet, wird er aus der Routing-Tabelle entfernt.

4.3 Verbindungsmanagement

Das Verbindungsmanagement ist ein essentieller Bestandteil des Protokolls, da es die Grundlage für die Kommunikation zwischen den Teilnehmern bildet. Es ist dafür verantwortlich, dass die Nachrichtenübertragung zwischen den Teilnehmern funktioniert. Dazu gehört der Verbindungsaufbau, die Nachrichtenübertragung und der Verbindungsabbau, welche im Folgenden detailliert beschrieben werden.

4.3.1 Verbindungsaufbau

Bei der Herstellung einer Verbindung zwischen zwei Teilnehmern sind die IP-Adresse und der Port des Zielgeräts entscheidend. Der Port ist durch die Definition in Abschnitt 4.2 bereits festgelegt und damit bekannt. Sobald die IP-Adresse des Ziels über Kademlia ermittelt wurden, wird versucht, eine direkte Verbindung herzustellen. Grundsätzlich gilt, dass eine direkte Verbindung vor allen anderen Verbindungsarten bevorzugt wird. Im Falle einer erfolglosen Suche im Kademlia-Netzwerk, gilt der Teilnehmer zu diesem Zeitpunkt als nicht erreichbar bzw. *offline*. Wird allerdings eine IP-Adresse ermittelt, wird ein UDP-Paket an diese gesendet. Es wird erwartet, dass das Ziel innerhalb eines festgelegten Zeitfensters antwortet. Wenn eine Antwort innerhalb dieses Zeitrahmens eingeht, wird die Kommunikation über diesen direkten Weg aufgebaut und ein Austausch von Nachrichten kann beginnen. Falls keine Antwort eintrifft, gibt es das zweite Verfahren, um eine Verbindung herzustellen: das *Interactive Connectivity Establishment* (kurz: *ICE*) Protokoll. Wenn also die direkte Verbindung scheitert, tritt das ICE-Protokoll in Kraft. Ursprünglich war die Idee, dass bei einer fehlgeschlagenen direkten Verbindung ein Relay-Server mit dem TURN-Protokoll verwendet werden soll. Doch aus nachfolgendem Zitat ist zu entnehmen, dass TURN nicht die beste Lösung ist, da nicht definiert ist, wie die *relayed transport address*, mit deren Hilfe die Kommunikation über den Relay-Server aufgebaut werden soll, an die Teilnehmer kommuniziert werden soll.

> *A client using TURN must have some way to communicate the relayed transport address to its peers and to learn each peer's IP address and port (more precisely, each peer's server-reflexive transport address; see Section 3). How this is done is out of the scope of the TURN protocol. One way this might be done is for the client and peers to exchange email messages.* (Reddy u. a. 2020, S. 7)

Deshalb wurde sich, wenn die direkte Verbindung nicht funktioniert, für die Verwendung des ICE-Protokolls entschieden. ICE ist ein Protokoll, das die Verbindungseinrichtung

zwischen zwei Geräten ermöglicht, die sich hinter NATs (Network Address Translation) befinden (Keränen, Holmberg und Rosenberg 2018, S. 6-7). Es ist ein Protokoll, das in der Lage ist, sich an sich ändernde Netzwerkbedingungen anzupassen und verschiedene Arten von Verbindungen zu unterstützen, wie beispielsweise direkte Verbindungen, Verbindungen über STUN (Session Traversal Utilities for NAT) oder Verbindungen über TURN (Traversal Using Relays around NAT) (Keränen, Holmberg und Rosenberg 2018, S. 1).

Um eine Verbindung zwischen zwei Geräten aufzubauen, durchläuft ICE drei Phasen: die Sammlung und der Austausch von Verbindungsadressen, die Konnektivitätsprüfung und die Auswahl der am besten geeigneten Verbindungsadresse für die Kommunikation (Keränen, Holmberg und Rosenberg 2018).

Die erste Phase beinhaltet das Sammeln verschiedener potenzieller Verbindungsadressen (auch *Kandidatenadressen* genannt (Keränen, Holmberg und Rosenberg 2018, S. 8)), die für eine zuverlässige Kommunikation benötigt werden, insbesondere wenn einer oder beide Teilnehmer sich hinter NATs befinden. Zuallererst werden die lokalen oder *Host*-Adressen der beteiligten Geräte berücksichtigt. Diese Adressen repräsentieren die standardmäßigen IP-Adressen und Ports der Geräte innerhalb ihres lokalen Netzwerks. Sie dienen als potenzielle direkte Verbindungswege zwischen den Geräten, falls sie sich im gleichen Netzwerk oder Subnetz befinden. Zusätzlich zu den lokalen Adressen werden reflexive Adressen über STUN (Session Traversal Utilities for NAT) ermittelt. STUN ermöglicht es einem Gerät, seine eigene externe IP-Adresse und den entsprechenden Port zu identifizieren, wie sie von einem NAT-Gerät reflektiert werden (Petit-Huguenin u. a. 2020, S. 4). Diese reflexiven Adressen stellen die externe Sichtbarkeit des Geräts aus der Perspektive des NAT-Geräts dar und helfen bei der Umgehung von NAT-Beschränkungen für den direkten Verbindungsaufbau. Sollte es jedoch aufgrund von restriktiven NAT-Konfigurationen nicht möglich sein, eine direkte Verbindung aufzubauen, kommen Relay-Adressen durch TURN (Traversal Using Relays around NAT) ins Spiel. Durch TURN werden Relay-Server genutzt, um den Datenverkehr zwischen den Geräten zu vermitteln (Reddy u. a. 2020, S. 10-11). Diese Weiterleitungsadressen dienen als alternative Verbindungsmethode, indem sie den Datenverkehr über den Relay-Server leiten und so die Hindernisse von restriktiven NATs umgehen. Die kombinierte Nutzung dieser verschiedenen Arten von potenziellen Verbindungsadressen – von lokalen, reflexiven bis hin zu Relay-Adressen – ermöglicht eine Vielzahl von Optionen für die Verbindungseinrichtung zwischen den Geräten. Diese Vielfalt an Adressen gewährleistet, dass selbst in komplexen Netzwerkszenarien wie NATs oder Firewalls verschiedene Wege für eine zuverlässige Kommunikation vorhanden sind.

Nachdem alle potenziellen Verbindungsadressen gesammelt wurden, werden sie an den anderen Teilnehmer gesendet. Dieser Prozess wird als *Kandidatenaustausch* bezeichnet und erfolgt unter Verwendung eines Signaling Servers. Aus der Definition von ICE geht nicht hervor, wie die Verbindungsadressen zwischen den Teilnehmern (in ICE als *Agents* bezeichnet) ausgetauscht werden sollen. Wie aus dem nachstehenden Zitat zu verstehen ist, nimmt das Protokoll an, dass die Agents dazu in der Lage sind, doch wie genau das geschehen soll, ist nicht definiert (Keränen, Holmberg und Rosenberg 2018).

> *In a typical ICE deployment, there are two endpoints (ICE agents) that want to communicate. [...]. ICE assumes that the agents are able to establish a signaling connection between each other.* (Keränen, Holmberg und Rosenberg 2018, S. 7)

Eine eigene Definition für den Austausch der Verbindungsadressen zwischen den Teilnehmern über einen Signaling Server zu erstellen, ginge über den Umfang dieser Arbeit hinaus. Ein Ansatz wäre, dass der Server die Benutzernamen der Teilnehmer kennt oder die Existenz des Benutzernamens eventuell via Smart Contract prüfen kann (siehe 4.6.4 *Registrierung*), und somit die Verbindungsadressen der anfragenden Teilnehmer anhand der Benutzernamen an die jeweiligen Teilnehmer sendet. Abbildung 4.4 stellt ein Beispiel eines Ablaufs des Kandidatenaustauschs über einen Signaling Server mit Hilfe eines Sequenzdiagramms dar:

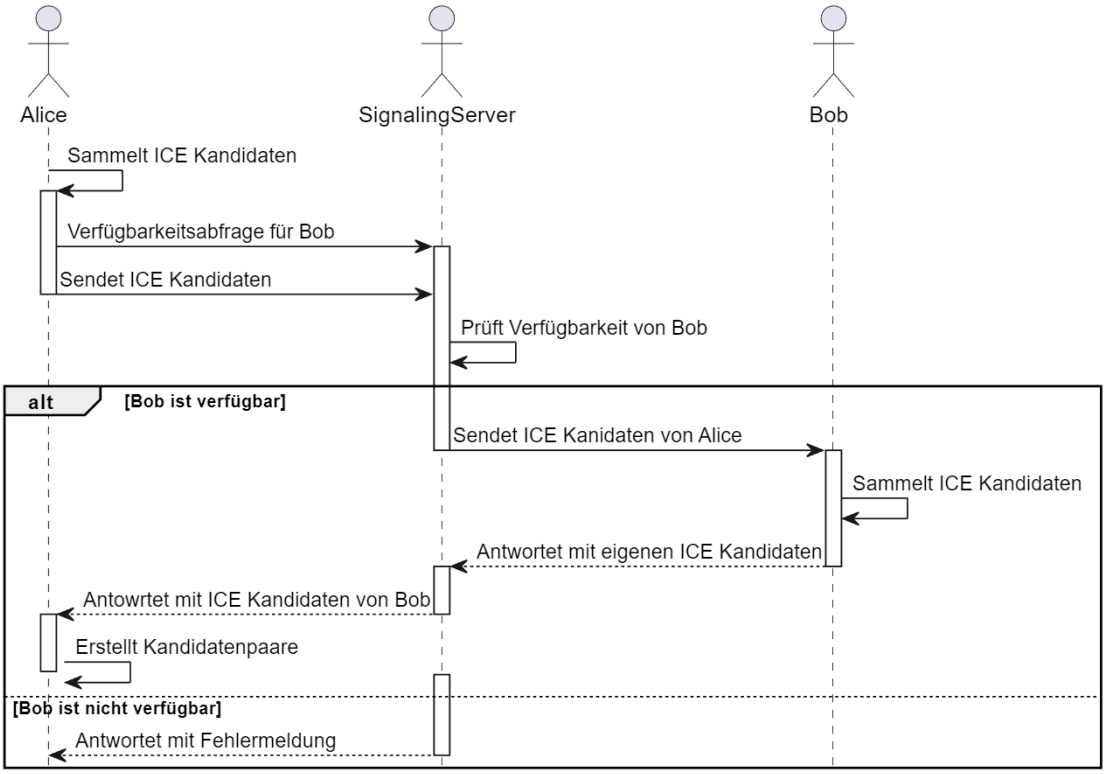

Abbildung 4.4: Kandidatenaustausch über einen Signaling Server

Alice möchte eine Verbindung zu Bob aufbauen. Dazu sendet sie eine Verfügbarkeitsabfrage an den Signaling Server. Zusätzlich sendet sie ihre Kandidatenadressen mit, die sie zuvor gesammelt hat. Der Signaling Server erhält die Kandidatenadressen von Alice und prüft, ob Bob verfügbar ist. Wenn dies der Fall ist, sendet der Signaling Server die Kandidatenadressen von Alice an Bob. Bob speichert die Kandidatenadressen von Alice und sendet seine eigenen Kandidatenadressen an den Signaling Server. Der Signaling Server leitet diese dann an Alice weiter. Sollte Bob allerdings nicht verfügbar sein, sendet der Signaling Server eine Fehlermeldung an Alice.

Man könnte nun argumentieren, dass Kademlia durch die Verwendung von ICE überflüssig wird, da auch ICE die lokale IP-Adresse und den Port des Empfängers ermitteln kann und somit eine direkte Verbindung aufgebaut werden könnte. Doch Kademlia wird bewusst weiterhin verwendet, da es die Auslastung des Signaling Servers so gering wie möglich hält, indem der erste Versuch des Verbindungsaufbaus immer über Kademlia erfolgt. Außerdem ist immer noch das Ziel, möglichst ohne jegliche Server auszukommen. Aus diesem Grund wird das ICE-Protokoll nur dann verwendet, wenn der Verbindungs-

aufbau über Kademlia fehlschlägt. Durch das Beibehalten dieser Struktur wird die Anzahl der Verbindungsversuche über den Signaling Server minimiert. Je nach Teilnehmeranzahl kann die Anzahl der Verbindungsversuche über den Signaling Server sehr hoch sein, was zu einer hohen Auslastung des Servers führen kann.

In Phase zwei werden Konnektivitätsprüfungen durchgeführt. Diese Prüfungen dienen dazu, die Eignung und Zuverlässigkeit der gesammelten Verbindungswege zwischen den Geräten zu bewerten, um die bestmögliche Verbindung für eine erfolgreiche Kommunikation zu identifizieren. Die Reihenfolge der Konnektivitätsprüfungen wird durch einen Prioritätsalgorithmus bestimmt, der die Verbindungsadressen nach ihrer Priorität ordnet. Dies wird vom Initiator der Verbindung durchgeführt, der die Verbindungsadressen der anderen Teilnehmer erhält. Aus der Dokumentation von ICE geht hervor, dass die Priorität einer Verbindungsadresse durch die folgende Formel berechnet wird (Keränen, Holmberg und Rosenberg 2018, S. 22):

$$\text{priority} = 2^{24}\cdot(\text{type preference})+2^{8}\cdot(\text{local preference})+2^{0}\cdot(256 - \text{component ID}) \quad (4.1)$$

Jeder Kandidat hat eine Priorität, die durch die Formel 4.1 berechnet wird. Die Priorität wird durch die drei Parameter in der Formel bestimmt: Typpräferenz, lokale Präferenz und Komponenten-ID. Die Typpräferenz ist ein Parameter, der die Art der Verbindungsadresse angibt, und dessen Wert sich zwischen 0 und 126 befinden muss. Es gibt, wie bereits in Phase eins aufgezeigt, drei verschiedene Typen von Verbindungsadressen: Host-Adressen, Peer-reflexive Adressen und Relay-Adressen. Die Werte für die Typpräferenz, die in der Dokumentation von ICE für die Berechnung der Priorität empfohlen werden, sind die folgenden: 126 für Host-Adressen, 100 für Peer-reflexive Adressen und 0 für Relay-Adressen. Wichtig ist, dass der Wert 0 für Relay-Adressen nicht bedeutet, dass Relay-Adressen nicht verwendet werden sollten, sondern dass sie die niedrigste Priorität haben. Für das Protokoll dieser Arbeit werden diese Werte übernommen. Für die lokalen Präferenzen wird ein Wert von 65535 empfohlen, um die Verwendung von lokalen Adressen zu priorisieren. Auch dieser Wert wird übernommen. Der dritte und letzte Wert der Formel ist die Komponenten-ID, die dazu dient, verschiedene Datenströme oder Komponenten innerhalb eines einzelnen Kandidaten zu unterscheiden. Das bedeutet, dass ein Kandidat mehrere Komponenten haben kann, die jeweils eine eigene Komponenten-ID haben. Dadurch können beispielweise Bild-Daten, Audio-Daten und Text-Daten innerhalb eines einzelnen Kandidaten über verschiedene Komponenten gesendet werden. Da in dem Protokoll dieser Arbeit nur ein Datenstrom verwendet wird - und zwar Text - wird

der Komponenten-ID der Wert 1 zugewiesen. Zur Veranschaulichung wird die Berechnung der Priorität für die drei verschiedenen Arten von Adressen anhand der vergebenen Werte in der Formel 4.1 dargestellt. Für die Berechnung der Priorität für Host-Adressen sieht die Formel wie folgt aus:

$$\text{priority} = 2^{24} \cdot (126) + 2^{8} \cdot (65535) + 2^{0} \cdot (256 - 1) \tag{4.2}$$

Die Priorität für Peer-reflexive Adressen lautet wie folgt:

$$\text{priority} = 2^{24} \cdot (100) + 2^{8} \cdot (65535) + 2^{0} \cdot (256 - 1) \tag{4.3}$$

Die Priorität für Relay-Adressen lautet wie folgt:

$$\text{priority} = 2^{24} \cdot (0) + 2^{8} \cdot (65535) + 2^{0} \cdot (256 - 1) \tag{4.4}$$

Wenn alle Kandidaten ihre Priorität erhalten haben, werden sie nach ihrer Priorität sortiert und die Konnektivitätsprüfung beginnt. Dieser Prozess beinhaltet den Versuch, Verbindungen aufzubauen und Datenverkehr über verschiedene potenzielle Wege zu senden und zu empfangen. Durch das Senden von Probe-Paketen über jede potenzielle Verbindungsadresse wird geprüft, ob eine Verbindung aufgebaut werden konnte. Dabei werden die drei verschiedenen Arten von Adressen (lokale, reflexive und Relay-Adressen) verwendet, um verschiedene Möglichkeiten zu testen, wie die Geräte miteinander kommunizieren können.

Die Probe-Pakete werden über UDP (kurz für: *User Datagram Protocol*) gesendet, da es ein verbindungsloses Protokoll ist und somit vor dem Senden keine Verbindung aufgebaut werden muss (Postel 1980, S. 1). Dies ermöglicht es, die Erreichbarkeit der Verbindungsadressen zu testen, ohne eine Verbindung aufzubauen. Die Probe-Pakete werden an die Verbindungsadressen gesendet und die Antwort wird überwacht. Wenn eine Antwort empfangen wird, wird die Verbindungsadresse als erreichbar angesehen. Wenn keine Antwort empfangen wird, wird die Verbindungsadresse als nicht erreichbar angesehen. Um die Stabilität der Verbindungsadressen zu testen, werden in regelmäßigen Abständen Probe-Pakete gesendet. Wenn die Probe-Pakete über einen längeren Zeitraum nicht beantwortet werden, wird die Verbindungsadresse als nicht stabil angesehen.

Mit dem Abschluss der Konnektivitätsprüfungen beginnt die dritte und letzte Phase. Mit den Ergebnissen werden Kandidaten-Paare gebildet, die die Verbindungsadressen der beiden Geräte enthalten. Während dieses Schritts werden die gesammelten Kandidaten (lokale IP-Adressen, Ports und durch STUN oder TURN-Server erhaltene Reflexionen)

in verschiedenen Konstellationen kombiniert. Jede dieser Kombinationen bildet ein Paar, das als möglicher Kommunikationsweg zwischen den Geräten dienen könnte. Die gebildeten Paare werden auf Redundanz kontrolliert und dann nach ihrer Priorität sortiert, die aus den einzelnen Prioritäten der Kandidaten berechnet wird. Aus dieser Reihenfolge wird dann die sogenannte *Checkliste* erstellt, die die Paare in der Reihenfolge ihrer Priorität enthält. Auch die Kandidaten-Paare erhalten mit Hilfe einer Formel eine Priorität, die in der Dokumentation von ICE auf Seite 31 definiert ist (Keränen, Holmberg und Rosenberg 2018, S. 31). Anschließend wird damit begonnen, eine Verbindung über das Paar mit der höchsten Priorität aufzubauen. Wenn die Verbindung erfolgreich aufgebaut werden kann, wird die Kommunikation über dieses Paar fortgesetzt. Wenn die Verbindung nicht erfolgreich aufgebaut werden kann, wird das Paar mit der nächsthöheren Priorität ausgewählt und der Prozess wird wiederholt. Dieser Prozess wird fortgesetzt, bis eine Verbindung erfolgreich aufgebaut werden kann. Wenn keine Verbindung aufgebaut werden kann, wird die Kommunikation über das Paar mit der höchsten Priorität fortgesetzt, das eine Verbindung über den Relay-Server verwendet. Die folgende Abbildung 4.5 stellt den Ablauf des ICE-Prozesses bildlich, mit Hilfe eines Sequenzdiagramms, dar:

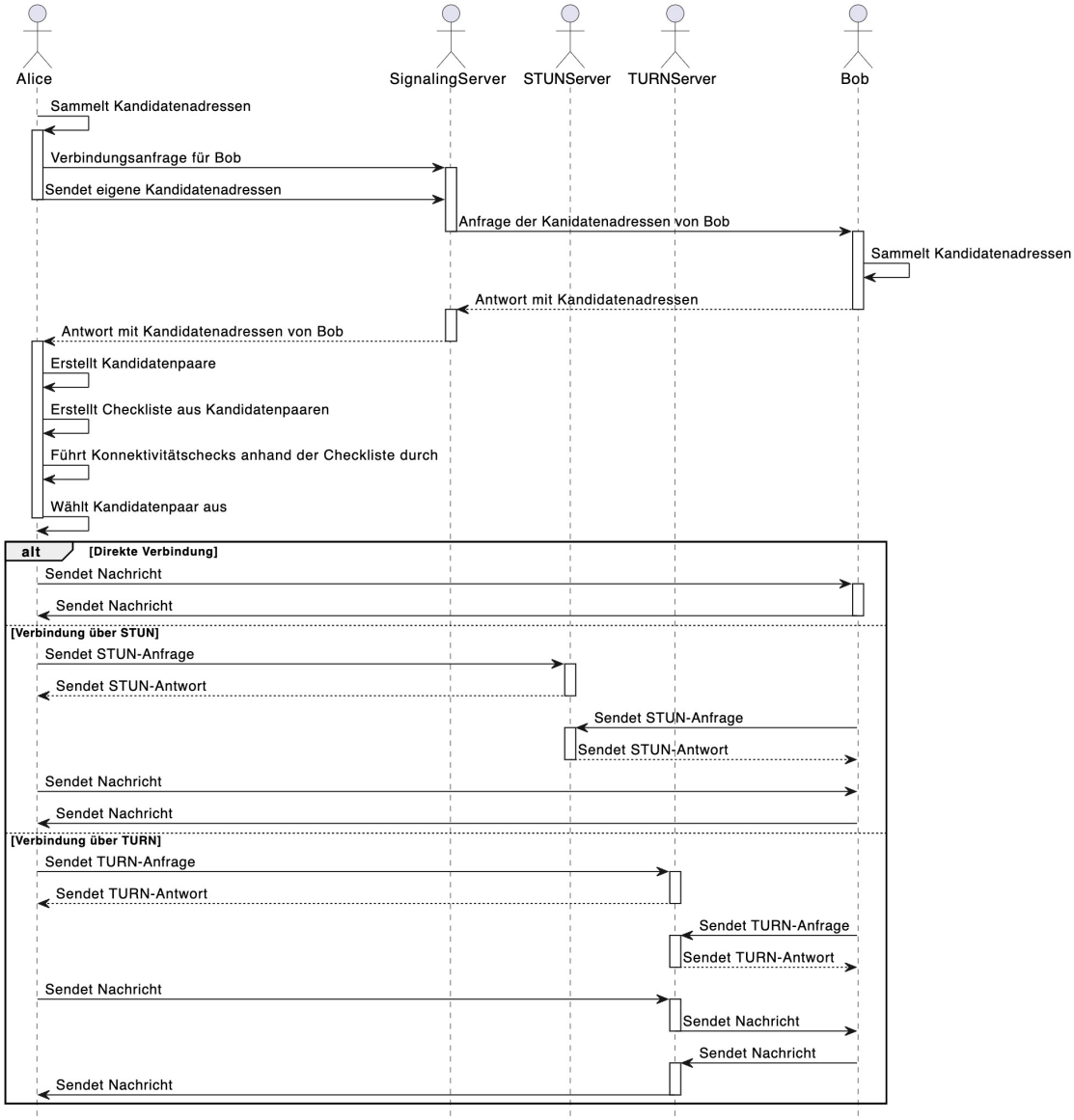

Abbildung 4.5: ICE-Prozess

Ein großer Vorteil von ICE ist die Flexibilität, die es ermöglicht, sich an sich ändernde Netzwerkbedingungen anzupassen. Falls sich während der Kommunikation Netzwerkparameter ändern - beispielsweise durch einen Wechsel zwischen Wi-Fi und Mobilfunknetzwerken - kann ICE dynamisch neue Kandidatenadressen identifizieren und die Verbindungen anpassen, ohne die laufende Kommunikation zu unterbrechen.

4.3.2 Nachrichtenübertragung

Wenn die am besten geeignete Verbindungsadresse eine lokale Adresse ist, wird eine direkte Verbindung zwischen den Geräten aufgebaut. Wenn die am besten geeignete Verbindungsadresse eine reflexive Adresse ist, wird eine direkte Verbindung zwischen den Geräten aufgebaut, indem die reflexive Adresse als Zieladresse verwendet wird. Wenn wiederum die am besten geeignete Verbindungsadresse eine Relay-Adresse ist, wird eine Verbindung über den Relay-Server aufgebaut, indem die Relay-Adresse als Zieladresse verwendet wird. Der Nachrichtenaustausch erfolgt über die ausgewählte Verbindungsadresse. Wobei hier mit *Verbindung aufbauen* gemeint ist, dass die Nachrichten über die Verbindungsadresse gesendet und empfangen werden können. Aus Sicht der Transportschicht wird durch die Verwendung von UDP keine Verbindung aufgebaut, denn UDP ist, wie bereits erwähnt, ein verbindungsloses Protokoll. Das bedeutet, dass keine Verbindung aufgebaut werden muss, um die Nachrichten zu senden und zu empfangen. Aus Applikationssicht wird jedoch eine Verbindung aufgebaut, da die Nachrichten zwischen den Teilnehmern ausgetauscht werden können.

Ein Nachteil bei der Verwendung von UDP ist, dass die Nachrichten nicht zuverlässig zugestellt werden. Das bedeutet, dass Nachrichten verloren gehen können, ohne dass der Absender oder Empfänger davon erfährt. Das Transportprotokoll TCP (Transmission Control Protocol) hingegen ist zuverlässig, da es eine Verbindung aufbaut und sicherstellt, dass die Nachrichten erfolgreich zugestellt werden (Eddy 2022, S. 36). Im Falle von Paketverlusten ist auf der gleichen Seite definiert, dass TCP in der Lage ist, die verlorenen Pakete zu erkennen und erneut zu senden.

Doch eine mögliche Verwendung von TCP hat auch Nachteile: ein Nachteil ist, dass es eine Verbindung aufbauen muss, bevor die Nachrichten gesendet werden können. Dies kann zu einer höheren Latenz führen. Ein weiterer Nachteil ist, dass die Verbindung aufrechterhalten werden muss, um die Nachrichten zu senden und zu empfangen. Dies führt zu einem höheren Ressourcenverbrauch, da die Verbindung aufrechterhalten werden muss, auch wenn keine Nachrichten gesendet werden. Auch bei NATs hat TCP Nachteile. NATs schließen die Verbindungen nach einer gewissen Zeit, wenn keine Daten übertragen werden. Dies kann dazu führen, dass die Verbindung geschlossen werden, bevor die Nachrichten gesendet werden können. Ein überzeugendes Argument für die Verwendung von UDP liegt in seiner Effizienz durch den geringeren Overhead im Vergleich zu TCP. Dank des schlankeren Headers werden weniger Daten übertragen, was besonders in Instant Messaging-Szenarien, in denen kleine Nachrichten häufig versendet werden, zu einer effizienteren Nutzung der Netzwerkressourcen und zu schnelleren Übertragungen führt.

UDP bietet zudem den Vorteil der Unabhängigkeit in Bezug auf die Reihenfolge der übermittelten Pakete. Im Gegensatz zu TCP, das die korrekte Reihenfolge sicherstellt, überlässt UDP diese Aufgabe der Anwendungsschicht. In Instant-Messaging-Anwendungen, in denen die Reihenfolge der Nachrichten möglicherweise weniger kritisch ist, ermöglicht dies eine flexiblere und potenziell schnellere Übertragung von Nachrichten. Die geringere Verbindungsetablierungszeit von UDP ist ein weiterer Pluspunkt. Da UDP ein verbindungsloses Protokoll ist, entfällt der zeitliche Aufwand für das Auf- und Abbauen von Verbindungen. Dies ist besonders vorteilhaft in Umgebungen mit häufigen Verbindungswechseln, wie sie bei Instant-Messaging mittels Peer-to-Peer auftreten können. Die Minimierung der Latenzzeit trägt dazu bei, Nachrichten schneller bereitzustellen (Postel 1980; Eddy 2022). Aus diesen Gründen wurde UDP als Transportprotokoll für die Nachrichtenübertragung gewählt.

4.3.3 Verbindungsabbau

Durch die Verwendung von UDP ist es nicht möglich, eine Verbindung aktiv zu schließen. Deshalb wird die Verbindung durch einen Timeout geschlossen, der nach einer gewissen Zeit abläuft, wenn keine Nachrichten mehr gesendet werden.

4.4 Nachrichtenformat

Jedes Instant-Messaging-Protokoll benötigt die Definition eines Nachrichtenformats, das die Struktur der Nachrichten festlegt, die zwischen den Teilnehmern ausgetauscht werden. Im Folgenden wird das Nachrichtenformat für das Instant-Messaging-Protokoll beschrieben.

4.4.1 Verbindungsanfrage

Um eine Verbindung zwischen zwei Teilnehmern herzustellen, muss zunächst eine Anfrage gesendet werden. Diese Anfrage enthält die folgenden Informationen:

- Nachrichtenformat: immer *0x01*
- Benutzername des Senders
- Benutzername des Empfängers
- Timestamp

- Signatur

Das Nachrichtenformat ist eine binäre Zahl, die angibt, dass es sich um eine Anfrage handelt. Die Benutzernamen erlauben die Zuordnung der Anfrage. Der Timestamp enthält die aktuelle Zeit, zu der die Anfrage gesendet wird. Die Signatur dient der Authentifizierung des Senders (siehe 4.5 *Vertrauliche Kommunikation*). Aus Sicherheitsgründen wird jede Anfrage signiert.

4.4.2 Nachrichtenaustausch

Nachdem eine Verbindung zwischen zwei Teilnehmern hergestellt werden konnte, kann die Nachricht vom Sender an den Empfänger gesendet werden. Die Nachricht enthält die folgenden Informationen:

- Nachrichtenformat: immer *0x02*
- Benutzername des Senders
- Benutzername des Empfängers
- Timestamp
- Signatur
- Nachrichtenlänge
- Nachrichteninhalt

Bei der Erstellung des Nachrichtenformats wurde sich am Vorschlag von Day et al. der Etablierung eines Standards orientiert, der in RFC 2779 mit dem Titel *Instant Messaging/Presence Protocol Requirements* definiert ist (Day u. a. 2000, S. 9).

Die ersten fünf Felder sind identisch mit denen der Verbindungsanfrage. Hinzu kommt die Nachrichtenlänge, die die Länge des eigentlichen Nachrichteninhalts angibt und der Nachrichteninhalt, der die eigentliche Nachricht enthält, die vom Sender an den Empfänger gesendet werden soll.

4.5 Vertrauliche Kommunikation

Die Kommunikation zwischen den Teilnehmern soll vertraulich sein. Das bedeutet, dass die Nachrichten nur von den Teilnehmern gelesen werden können, die an der Kom-

munikation beteiligt sind. Um dies gewährleisten zu können, wird eine Ende-zu-Ende-Verschlüsselung verwendet. Das in Abschnitt 2.3.4 *Ende-zu-Ende-Verschlüsselung* besprochene Signal-Protokoll ist momentan der Industriestandard für Ende-zu-Ende-Verschlüsselung und wird von den bekanntesten Messengern verwendet (Wong 2023, S. 260).

Deshalb soll dieses Protokoll auch für die Sicherheit der Kommunikation in dieser Arbeit verwendet werden. Allerdings ist das Signal-Protokoll auf eine Client-Server-Architektur ausgelegt. Da in dieser Arbeit eine Peer-to-Peer-Architektur verwendet wird, muss das Signal-Protokoll angepasst werden.

Der Double Ratchet Algorithmus (siehe 2.3.4 *Ende-zu-Ende-Verschlüsselung*) wird übernommen. Damit wird die *Forward Secrecy* gewährleistet. Der X3DH-Schlüsselaustausch kann allerdings nicht verwendet werden.

4.5.1 Schlüsselvereinbarung

Für die Implementierung eines Peer-to-Peer-Schlüsselaustauschs kamen zwei Verfahren in Frage. Zum einen der *Diffie-Hellman-(DH-)*Schlüsselaustausch und zum anderen der *Elliptic Curve Diffie-Hellman-(ECDH-)*Schlüsselaustausch. Grundsätzlich baut der Diffie-Hellman-Schlüsselaustausch auf das mathematische Gebiet der *Gruppentheorie* auf. Der DH-Schlüsselaustausch ist ein Schlüsselaustauschverfahren, das auf dem diskreten Logarithmusproblem basiert. Der ECDH-Schlüsselaustausch hingegen basiert auf elliptischen Kurven. David Wong empfiehlt in seinem Buch *Kryptografie in der Praxis* ECDH zu verwenden, da die Schlüssel kleiner sind und noch keine starken Angriffe gegen das Verfahren gefunden wurden (Wong 2023, S. 101-125).

Aus diesem Grund wurde sich für den ECDH-Schlüsselaustausch entschieden. Hierfür muss eine elliptische Kurve definiert werden. Wong zählt hierfür zwei Kurven auf, die von den meisten Anwendungen verwendet werden: *P-256* und *Curve25519*. Da die Erzeugung der Kurve P-256 unklar ist und damit die Vertrauenswürdigkeit darunter leidet, wurde sich für Curve25519 entschieden (Wong 2023, S. 121). Den ECDH-Schlüsselaustausch mit Curve25519 nennt man auch X25519 und wird in Langley, Hamburg und Turner 2016 spezifiziert.

Solch ein Schlüsselaustauschverfahren für sich genommen, hat den Nachteil, dass ein aktiver Angreifer die Kommunikation zwischen den Teilnehmern abhören und manipulieren kann. Um dies zu verhindern, wird ein *authentifizierter* Schlüsselaustausch verwendet. Dieser setzt sich zusammen aus dem Schlüsselaustauschverfahren und einer Signatur. Die Signatur wird mit dem privaten statischen Schlüssel des Senders erstellt und kann mit dem öffentlichen statischen Schlüssel des Senders verifiziert werden. Die Wahl eines Si-

gnaturverfahrens ist nicht trivial. Wong zählt zwei moderne Verfahren auf: *Elliptic Curve Digital Signature-Algorithmus (ECDSA)* und *Edwards-curve Digital Signature Algorithm (EdDSA)*. Dabei erwähnt er auch Vertrauensbedenken gegenüber ECDSA, weshalb sich für EdDSA entschieden wurde. EdDSA ist ein Signaturverfahren, das auf elliptischen Kurven basiert, weshalb auch hier eine Kurve ausgewählt werden muss. Wong beschreibt, dass in der Praxis meist die Kurve *Edwards25519* verwendet wird. Diese Kombination nennt man auch Ed25519 und wird in Josefsson und Liusvaara 2017 spezifiziert (Wong 2023, S. 160-172).

Das modifizierte Signal-Protokoll ist in Abbildung 4.6 dargestellt.

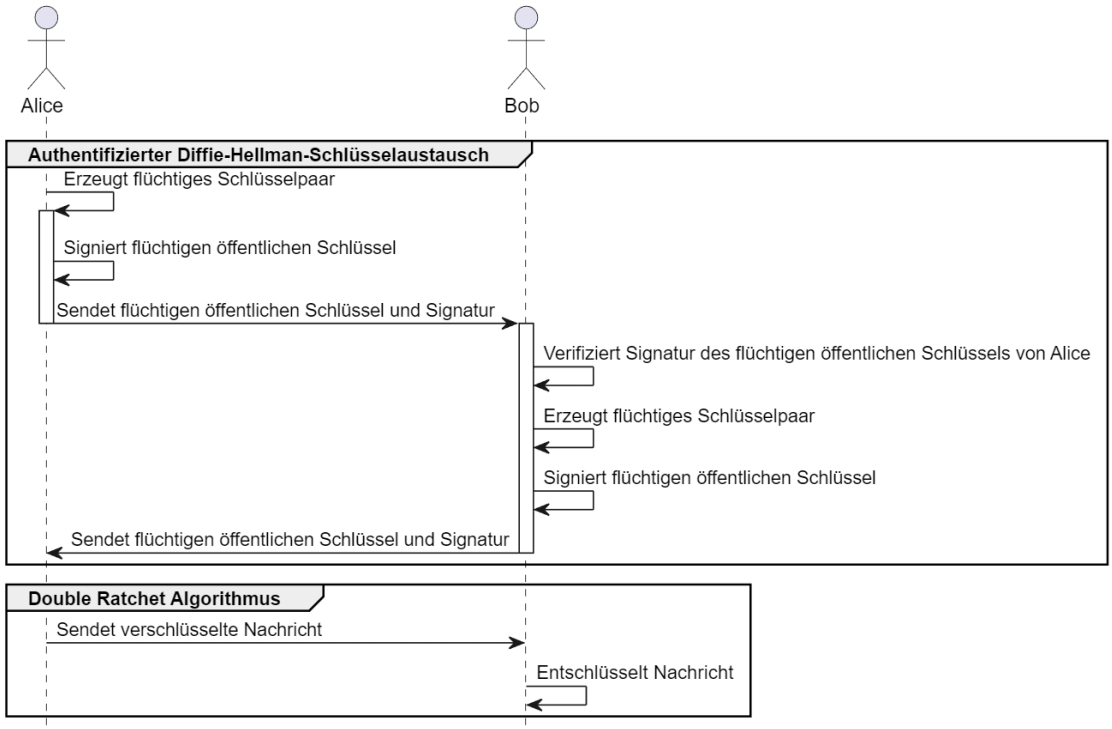

Abbildung 4.6: Modifizierte Ende-zu-Ende-Verschlüsselung

Im Block *Authentifizierter Diffie-Hellman Schlüsselaustausch* aus Abbildung 4.6 erzeugt Alice ein neues X25519-Schlüsselpaar und signiert ihren öffentlichen Schlüssel unter Verwendung des privaten statischen Ed25519-Schlüssels. Dieser signierte öffentliche Schlüssel wird an Bob gesendet. Bob erzeugt ebenfalls ein X25519-Schlüsselpaar und signiert seinen öffentlichen Schlüssel mit seinem privaten statischen Ed25519-Schlüssel. Dieser signierte öffentliche Schlüssel wird an Alice gesendet. Alice und Bob können nun das gemeinsame Geheimnis berechnen (siehe 2.4 *Schlüsselvereinbarung (in Anlehnung an Wong 2023, S.*

102)) und damit den Double Ratchet Algorithmus initialisieren.

4.5.2 Statische Schlüssel

Die statischen Schlüssel werden verwendet, um die Teilnehmer zu authentifizieren (Berechnung und Verifikation der Signaturen). Diese Schlüssel werden einmalig bei der Registrierung (siehe 4.2 *Peer-Discovery und Routing*) eines Teilnehmers erzeugt. Der private Schlüssel wird lokal auf dem Gerät des Teilnehmers gespeichert. Für den öffentlichen Schlüssel muss ein Weg gefunden werden, diesen allen anderen Teilnehmern öffentlich zugänglich und eindeutig zuordenbar zu machen. Hierfür wird die Blockchain verwendet.

4.6 Sicherheit durch Blockchain

In diesem Abschnitt wird die Integration der Blockchain in das Protokoll beschrieben. Dazu werden zunächst die Möglichkeiten der Integration erläutert und ausgewählt. Darauf folgt die Beschreibung der ausgewählten Integrationsmöglichkeit und die dafür benötigten Funktionen, die die Blockchain bieten muss. Basierend darauf wird eine geeignete Blockchain ausgewählt und die Integration in das Protokoll beschrieben.

4.6.1 Möglichkeiten der Integration

Eine Blockchainintegration in das Protokoll kann auf unterschiedliche Weise erfolgen und verschiedene Vorteile bieten.

1. Eine Möglichkeit wäre die Verwendung der Blockchain zur Verwaltung von Benutzeridentitäten. Dazu wird jeder Teilnehmer auf der Blockchain registriert und erhält eine eindeutige ID. Diese ID kann dann als ID im Kademlia-Netzwerk verwendet werden. Dadurch würde die Blockchain als zentrale Instanz für die Verwaltung der Benutzeridentitäten dienen.

2. Eine weitere Anwendungsmöglichkeit könnte den Fokus auf Datenschutz und Anonymität legen. Hierfür könnten Smart Contracts auf der Blockchain verwendet werden, die sicherstellen, dass nur autorisierte Nutzer auf bestimmte Informationen zugreifen können. So könnte beispielsweise ein Smart Contract auf der Blockchain erstellt werden, der personenbezogene Daten wie den Namen, die Adresse oder die Telefonnummer eines Nutzers enthält. Dieser Smart Contract könnte dann so konfiguriert werden, dass nur der Nutzer selbst und bestimmte andere Nutzer,

die der Nutzer autorisiert hat, auf diese Daten zugreifen können. Dadurch würde die Blockchain als zentrale Instanz für die Verwaltung der Zugriffsrechte auf personenbezogene Daten dienen.

3. Die Blockchain könnte auch zur Verifizierung von Nachrichtenquellen verwendet werden. Dazu könnte jeder Teilnehmer des Protokolls auf der Blockchain registriert werden und einen öffentlichen Schlüssel hinterlegen. Dieser öffentliche Schlüssel könnte dann von anderen Teilnehmern verwendet werden, um die Identität des Absenders einer Nachricht zu verifizieren. Dadurch würde die Blockchain als zentrale Instanz für die Verwaltung der öffentlichen Schlüssel dienen.

4. Eine Protokollierung von Nachrichten wäre ebenfalls möglich. Dazu könnte jede Nachricht auf der Blockchain protokolliert werden. Dadurch hätte man eine öffentliche und unveränderliche Historie aller Nachrichten, die über das Protokoll gesendet wurden.

4.6.2 Integration in das Protokoll

Für die Auswahl der Integrationsmöglichkeit wurden die in Abschnitt 3 (*Anforderungsanalyse*) definierten funktionalen und nicht-funktionalen Anforderungen herangezogen. Die meisten funktionalen und nicht-funktionalen Anforderungen werden bereits durch die Verwendung von Kademlia und ICE erfüllt. Die Blockchain eignet sich wegen ihrer Manipulationssicherheit besonders für die Sicherstellung der Authentizität der Nachrichten (siehe Punkt 3). Damit einhergehend ist auch die Verwaltung der Benutzeridentitäten (siehe Punkt 1), da der gespeicherte öffentliche Schlüssel eindeutig einem Benutzer zugeordnet werden kann und somit die Identität des Benutzers eindeutig festgelegt wird. Die übrigen Integrationsmöglichkeiten (siehe Punkt 2 und 4) sind für die Anforderungen an das Protokoll nicht relevant und wurden daher nicht weiter betrachtet.

4.6.3 Auswahl der Blockchain

Für die Integration einer Blockchain in das Protokoll wurde zunächst eine geeignete Blockchain gesucht. Dabei wurde sich auf die beiden bekanntesten Blockchains, Bitcoin und Ethereum, beschränkt. Um eine Möglichkeit zu haben, eine Blockchain einzubinden, muss diese eine Funktion bieten, um beliebige Daten zu speichern und zu lesen. Dies kann durch Smart Contracts ermöglicht werden (siehe 2.4.6 *Smart Contracts*). Da Bitcoin eine reine Kryptowährung ist und keine Smart Contracts unterstützt, wurde sich für Ethereum entschieden.

Ethereum ist eine Blockchain, die zwar auch eine Kryptowährung names *Ether* besitzt, aber zusätzlich auch Smart Contracts unterstützt (siehe 2.4.5 *Ethereum*). Wenn ein Smart Contract ausgeführt wird, werden Ressourcen, wie Speicher und Rechenleistung, im Netzwerk verwendet. Die Knoten, die den Smart Contract ausführen, werden dafür mit Ether belohnt. Die Kosten für die Ausführung eines Smart Contracts werden in *Gas* gemessen. Gas ist eine separate virtuelle Währung, die die Kosten für die Ausführung eines Smart Contracts angibt. Es wird Gas statt Ether verwendet, da der Ether-Preis stark schwanken kann. Der Gas-Preis hingegen ist konstant und wird in *Gwei* gemessen. 1 Gwei entspricht dabei 10^{-9} Ether. Theoretisch wäre es möglich, den Preis für die Ausführung einer Methode innerhalb eines Smart Contracts auf 0 zu setzen. Allerdings würde dies zum einen dazu führen, dass ein Angreifer den Smart Contract unendlich oft ausführen könnte, was das Netzwerk überlasten könnte. Zum anderen würde es dazu führen, dass der Smart Contract nicht mehr ausgeführt wird, da die Knoten im Netzwerk keine Belohnung für die Ausführung erhalten und somit keinen Anreiz haben, den Smart Contract auszuführen (Antonopoulos und Wood 2018, S. 105-107).

Das bedeutet für die Teilnehmer des Protokolls, dass sie Ether besitzen müssen, um sich registrieren zu können. Für die zwei weiteren Funktionen, die die Blockchain im Protokoll übernimmt, ist keine Währung notwendig, da es sich um reine Lesezugriffe handelt, welche keine Kosten verursachen. Die beiden Funktionen sind die Suche nach einem Benutzernamen und die Suche nach einem öffentlichen Schlüssel anhand eines Benutzernamens. Die Blockchain wird also für die Registrierung der Teilnehmer und für die Suche nach Benutzernamen und öffentlichen Schlüsseln verwendet.

4.6.4 Registrierung

Um das Protokoll zu nutzen, muss sich jeder Nutzer zunächst auf der Blockchain registrieren. Dazu muss ein Smart Contract auf der Blockchain ausgeführt werden, der die Registrierung des Nutzers durchführt. Dieser Smart Contract wird mit dem Benutzernamen und dem statischen öffentlichen Schlüssel aufgerufen. Der statische öffentliche Schlüssel wird bei der Registrierung, bevor der Smart Contract angestoßen wird, festgelegt und kann nicht mehr geändert werden. Der Smart Contract erstellt einen neuen Eintrag in der Blockchain, der den Benutzernamen und den öffentlichen Schlüssel des Nutzers enthält. Um Dopplungen in der Blockchain zu vermeiden, wird der Benutzername nicht als Key in der Blockchain gespeichert, sondern als Value. Als Key wird die Adresse des Benutzers verwendet. Dadurch wird sichergestellt, dass jeder Benutzer nur einmal auf der Blockchain registriert werden kann. Diese Funktion wird nur einmalig

bei der Registrierung aufgerufen. Sollte ein Nutzer seinen Benutzernamen ändern wollen, muss er sich mit dem neuen Benutzernamen und dem öffentlichen Schlüssel eines neu erzeugten statischen Schlüsselpaars erneut registrieren. Das folgende Listing (4.1) zeigt die in Solidity geschriebene Funktion, die die Registrierung eines Nutzers auf der Blockchain durchführt.

```
// Function to register a user
function registerUser(string memory _username,
    bytes memory _publicKey) external {
        // Check if the username and public key are not empty
        require(bytes(_username).length > 0,
            "Username cannot be empty");
        require(_publicKey.length > 0,
            "Public key cannot be empty");

        // Check if the username is not already taken
        require(usernames[_username] == address(0),
            "Username is already taken");

        // Check if the user is not already registered
        require(!users[msg.sender].isRegistered, "User is already
            registered");

        // Add the user to the users mapping with the user's
        // username as the key
        users[msg.sender] = User(_username, _publicKey, true);

        // Store the username to address mapping
        usernames[_username] = msg.sender;

        // Emit the UserRegistered event with the user's address,
        // username and public key
        emit UserRegistered(msg.sender, _username, _publicKey);
}
```

Listing 4.1: Registrierung eines Nutzers auf der Blockchain

Für die Kommunikation mit anderen Teilnehmern, benötigt man die ID des anderen Teilnehmers im Kademlia-Netzwerk. Da die ID im Kademlia-Netzwerk dem Benutzernamen entspricht, muss der Benutzername des anderen Teilnehmers bekannt sein. Dazu wird dieser Benutzername auf der Blockchain gesucht, um zu kontrollieren, ob dieser bereits

registriert ist. Hierfür wird eine Funktion im Smart Contract auf der Blockchain aufgerufen, die den Benutzernamen als Parameter erhält. Der Smart Contract sucht dann in der Blockchain nach dem Benutzernamen und gibt ihn, falls vorhanden, zurück. Sollte kein Benutzer mit diesem Benutzernamen auf der Blockchain vorhanden sein, wird ein leerer String zurückgegeben, der darauf hinweist, dass der gesuchte Benutzer nicht registriert ist und somit keine Verbindung aufgebaut werden kann. Der Rückgabewert muss von der Applikation, die dieses Protokoll nutzt, abgefangen werden. Die Funktion könnte wie folgt aussehen:

```
// Function to check if a user is registered
function isUserRegistered(string memory _username) external view
    returns (bool) {
     // Try to get the address associated with the username
     address userAddress = usernames[_username];
     // Check if the address is empty
     if (userAddress == address(0)) {
          // If the address is empty, return false
          return false;
     }
     // If the address is not empty, get the user details using the
          address
     User storage user = users[userAddress];
     // Return the user's registration status
     return (user.isRegistered);
}
```

Listing 4.2: Suche nach einem Benutzernamen auf der Blockchain

4.6.5 Öffentlicher Schlüssel

Ist der Benutzername auf der Blockchain vorhanden und es kommt zum Verbindungsaufbau wie in Abschnitt 4.3 (Verbindungsmanagement) beschrieben, wird der öffentliche Schlüssel des Benutzers ausgelesen. Dafür wird die nachstehende Funktion genutzt:

```
// Function to get a user's public key
function getUserPublicKey(string memory _username) external view
    returns (bytes memory) {
          // Try to get the address associated with the username
          address userAddress = usernames[_username];

          // Check if the address is empty
```

```
8           require(userAddress != address(0),
9               "User does not exist.");
10
11          // If the address is not empty, get the user
12          // details using the address
13          User storage user = users[userAddress];
14
15          // Return the user's public key
16          return (user.publicKey);
17  }
```

Listing 4.3: Suche nach einem öffentlichen Schlüssel auf der Blockchain

Um Redundanz im Code zu vermeiden, wäre es sinnvoll das Suchen nach der Adresse des übergebenen Benutzernamens, die Prüfung ob die Adresse leer ist und das Auslesen der Benutzerdaten aus dem Mapping in eine eigene Funktion auszulagern, welche nur von innerhalb des Smart Contracts aufgerufen werden kann. Aus Gründen des besseren Lese- und Verständnisflusses wurde dies hier nicht umgesetzt.

Durch das Speichern des öffentlichen Schlüssels auf der Blockchain kann jeder Teilnehmer die öffentlichen Schlüssel der anderen Teilnehmer finden und die Nachrichten, die er erhält, mit dem öffentlichen Schlüssel des Absenders verifizieren. Dadurch kann sichergestellt werden, dass die Nachrichten tatsächlich vom angegebenen Absender stammen und nicht von einem anderen Teilnehmer gesendet wurden, der sich als jemand anderes ausgibt.

Kapitel 5

Evaluation

In diesem Kapitel wird das entwickelte Protokoll mit den in der Anforderungsanalyse definierten Anforderungen verglichen und Verbesserungsmöglichkeiten und Kritikpunkte aufgezeigt.

5.1 Erfüllung der Anforderungen

Dieser Abschnitt beinhaltet zuerst eine Betrachtung der funktionalen Anforderungen und anschließend eine Betrachtung der nicht-funktionalen Anforderungen. Dabei wird aufgezeigt, welche Anforderungen erfüllt wurden und welche nicht. Außerdem wird aufgezeigt, welche Anforderungen nur teilweise erfüllt wurden und warum dies der Fall ist.

5.1.1 Funktionale Anforderungen

Die funktionalen Anforderungen beziehen sich auf die Funktionalität des Protokolls und werden im Folgenden betrachtet.

Peer-Discovery and Routing

Das entwickelte Protokoll bietet die Möglichkeit, dass sich Teilnehmer anhand ihres Benutzernamens finden können. Die Anforderung der Peer-Discovery und des Routings wird deshalb als erfüllt angesehen. Eine Einschränkung ist, dass die Länge des Benutzernamens auf 20 Zeichen begrenzt ist. Das wäre lösbar, indem eine vom Benutzernamen unabhängige ID erzeugt und in der Blockchain abgelegt wird. Dadurch wäre der Benutzername frei wählbar und die ID wäre trotzdem eindeutig. Allerdings würde dies zusätzliche Transaktionen auf der Blockchain erfordern, die Geld kosten. Deshalb wurde diese Lösung nicht

implementiert.

Verbindungsmanagement

Auch diese Anforderung wird durch das entwickelte Protokoll erfüllt. Die Teilnehmer sind in der Lage eine Verbindung zu einem anderen Teilnehmer aufzubauen und Nachrichten auszutauschen. Hierbei wird ein mehrstufiges Verfahren verwendet, das die Herausforderungen eines reinen Peer-to-Peer-Netzwerks bezüglich NAT-Traversal (siehe Abschnitt 2.2.2 *Problemstellung und mögliche Lösungen*) löst.

Nachrichtenformatierung

Das Nachrichtenformat wurde so gestaltet, dass es die Anforderungen erfüllt. Es ist in Abschnitt 4.4 *Nachrichtenformat* beschrieben und kann somit von allen Teilnehmern verstanden werden.

Sicherheit und Verschlüsselung

Die Anforderungen an die Sicherheit und Verschlüsselung werden durch das entwickelte Protokoll teilweise erfüllt. Es wurde eine Ende-zu-Ende-Verschlüsselung implementiert, die die Vertraulichkeit der Nachrichten gewährleistet. Die Integrität der Nachrichten wird durch die Signatur gewährleistet. Für die Authentizität der Teilnehmer dient die Integration der Ethereum-Blockchain. Die Anforderungen an die Sicherheit des Netzwerks werden nur teilweise erfüllt, da keine Schutzmechanismen gegen Sybil-Angriffe oder Denial-of-Service-Angriffe implementiert wurden. Eine Möglichkeit, um Sybil-Angriffe zu erschweren, wäre das Verknüpfen der Kademlia-ID mit einer vom Benutzernamen unabhängigen ID, welche mit in die Blockchain geschrieben wird. Das würde zum einen verhindern, dass nicht-registrierte Nutzer dem Kademlia-Netzwerk beitreten können und zum anderen wäre die damit verpflichtende Registrierung mit Kosten verbunden, die ein Angreifer aufbringen muss, um am Netzwerk teilzunehmen. Da ein Sybil-Angriff darauf basiert, viele Identitäten zu verwenden, um das Netzwerk zu übernehmen, müsste der Angreifer für jede Identität die Kosten der Registrierung aufbringen. Dadurch wird der Angriff erschwert, da der Angreifer mehr Ressourcen benötigt, um das Netzwerk zu übernehmen. Was die Gefahr von Denial-of-Service-Angriffen angeht, so bietet Kademlia bereits einen gewissen Schutz gegen diese Angriffe, da durch die Verwendung der k-Buckets nur eine begrenzte Anzahl an Teilnehmern pro Knoten gespeichert werden kann. Hingegen bei ICE ist es möglich, dass ein Angreifer die ICE-Server (Signaling, STUN, TURN) mit Anfragen flutet, sodass dieser nicht mehr erreichbar ist. Um das zu

verhindern, müssten die Server bei jeder Anfrage prüfen, ob es sich um einen validen Teilnehmer handelt, was allerdings sehr aufwändig wäre. Eine andere Möglichkeit wäre, dass der Server die Anfragen an einen anderen Server weiterleitet, der die Anfragen verarbeitet. Dadurch würde der Server entlastet werden, allerdings würde die Anzahl der Server dadurch erhöht werden, was wiederum Kosten verursacht.

Plattformunabhängigkeit

Die Plattformunabhängigkeit wird durch die Verwendung von standardisierten Protokollen wie IP, UDP und ICE gewährleistet. Somit kann das entwickelte Protokoll auf allen Plattformen eingesetzt werden, die diese Protokolle unterstützen. Die Anforderung wird deshalb als erfüllt angesehen.

Fehlerbehandlung und Wiederholungsmechanismen

Die Anforderung an die Fehlerbehandlung und Wiederholungsmechanismen wird durch das entwickelte Protokoll nicht erfüllt. Durch die Verwendung von UDP als Transportprotokoll wird die Fehlerbehandlung nicht durch das Protokoll selbst übernommen, sondern müsste durch die Anwendung, die das Protokoll implementiert, übernommen werden. Einen Wiederholungsmechanismen, um verlorene Nachrichten erneut zu senden, fehlt ebenfalls. Die Anforderung wird deshalb als nicht erfüllt angesehen.

5.1.2 Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen beziehen sich auf die Eigenschaften des Protokolls und werden im Folgenden betrachtet.

Performanz

Die Anforderung an die Leistungsfähigkeit des Protokolls wird durch das entwickelte Protokoll erfüllt. Die Nachrichten werden durch das Transportprotokoll UDP schnell übertragen und die Verwendung von Kademlia als Routing-Algorithmus ermöglicht eine schnelle Suche nach Teilnehmern. Die Anforderung wird deshalb als erfüllt angesehen.

Sicherheit

Durch die Implementierung einer Ende-zu-Ende-Verschlüsselung wird die Vertraulichkeit der Nachrichten gewährleistet. Die in den Anforderungen definierten Sicherheitsziele

werden deshalb erfüllt und somit wird die Anforderung an die Sicherheit als erfüllt angesehen.

Zuverlässigkeit

Die Anforderung an die Zuverlässigkeit wird durch das entwickelte Protokoll nicht erfüllt. Da UDP als Transportprotokoll verwendet wird, werden Nachrichten nicht erneut gesendet und es wird auch nicht geprüft, ob eine Nachricht erfolgreich zugestellt werden konnte. Die Anforderung wird deshalb als nicht erfüllt betrachtet. Durch die Verwendung von TCP als Transportprotokoll könnte die Anforderung erfüllt werden, allerdings würde dies die Performanz des Protokolls, durch die zusätzlichen Überprüfungen, negativ beeinflussen.

Kompatibilität

Die Anforderung an die Kompatibilität wird durch das entwickelte Protokoll erfüllt. Es wurde sich an den RFCs orientiert, die die Protokolle definieren, die verwendet werden. Somit ist das entwickelte Protokoll kompatibel mit den RFCs und kann mit anderen Anwendungen, die diese Protokolle implementieren, kommunizieren. Die Anforderung wird deshalb als erfüllt betrachtet.

Skalierbarkeit

Was die Skalierbarkeit betrifft, wird die Anforderung durch das entwickelte Protokoll teilweise erfüllt. Durch die Verwendung von Kademlia als Routing-Algorithmus wird die Skalierbarkeit des Netzwerks gewährleistet. Doch durch ICE wird die Skalierbarkeit eingeschränkt, da die Anzahl der Teilnehmer, die sich gleichzeitig mit den ICE-Servern verbinden können, begrenzt ist. Die Anforderung wird deshalb als teilweise erfüllt betrachtet.

5.1.3 Verbesserungsmöglichkeiten und Kritikpunkte

In diesem Abschnitt werden Verbesserungsmöglichkeiten und Kritikpunkte für das entwickelte Protokoll aufgezeigt.

Kosten

Bei der Registrierung eines Nutzers fallen Kosten an, weil die Benutzerdaten durch die Verwendung eines Smart Contracts in die Blockchain geschrieben werden. Diese Kosten

könnten abschreckend sein, da es viele Alternativen gibt, die kostenlos sind und mehr Features bieten. Allerdings bietet die Verwendung der Blockchain den Vorteil, dass keiner zentralen Instanz vertraut werden muss, um die Authentizität der Gesprächspartner sicherzustellen.

Benutzernamen

Die Länge der Benutzernamen ist auf 20 Zeichen begrenzt, was für manche Benutzer nicht ausreichend sein könnte. Eine Möglichkeit, diese Einschränkung zu umgehen, wurde in Abschnitt 5.1.1 *Peer-Discovery and Routing* aufgezeigt.

Kapitel 6

Schlussfolgerung und Ausblick

In dieser Arbeit wurde ein Peer-to-Peer-Instant-Messaging-Protokoll entwickelt, das einen Schwerpunkt auf Sicherheit legt. Die Sicherheit wird durch die Verwendung von Ende-zu-Ende-Verschlüsselung und der Integration der Ethereum-Blockchain gewährleistet. Die Verwendung von Kademlia als Routing-Algorithmus ermöglicht eine schnelle Suche nach Teilnehmern und die Verwendung von ICE ermöglicht die Kommunikation zwischen Teilnehmern, die sich hinter NAT befinden.

Diese Arbeit hatte nicht den Anspruch eine vollständige Definition eines Protokolls zu liefern, sondern sollte einen Prototypen entwickeln, der die Machbarkeit eines solchen Protokolls aufzeigt. Deshalb wurden nicht alle Aspekte mit der eigentlich nötigen Tiefe beleuchtet. Es hat sich herausgestellt, dass die Verwendung von Peer-to-Peer-Mechanismen in Instant-Messaging-Protokollen durchaus sinnvoll, aber sehr komplex ist. Es müssen viele Aspekte beachtet werden, die in zentralisierten Protokollen nicht vorhanden sind, wie beispielsweise der Umgang mit NAT. Die Entwicklung einer Ende-zu-Ende-Verschlüsselung wurde durch die teilweise Verwendung des Signal-Protokolls vereinfacht. Die Auswahl der Algorithmen für die Schlüsselvereinbarung hat Potenzial zur Verbesserung, da kein genauer Vergleich mit anderen Algorithmen durchgeführt wurde. Die Integration der Blockchain ist jedoch eine sinnvolle Maßnahme, um die Sicherheit des Protokolls zu erhöhen, da keiner zentralen Instanz vertraut werden muss. Dies könnte die Akzeptanz des Protokolls erhöhen.

Eine prototypische Implementierung wäre ein sinnvoller nächster Schritt, um die Machbarkeit des Protokolls zu demonstrieren und eventuelle Definitionslücken zu schließen. Darüber hinaus könnten Metriken definiert und gemessen werden, um die Leistung des Protokolls zu bewerten.

Literatur

Alam, Shadab (2023). „The Current State of Blockchain Consensus Mechanism: Issues and Future Works“. In: *International Journal of Advanced Computer Science and Applications* 14.8, S. 84–94. DOI: 10.14569/IJACSA.2023.0140810. URL: http://dx.doi.org/10.14569/IJACSA.2023.0140810.

Antonopoulos, Andreas M. und Gavin Wood (2018). *Mastering Ethereum. Building Smart Contracts and DApps*. 1. Auflage. O´Reilly Media. ISBN: 978-1-491-97194-9.

Ayers, Rick, Sam Brothers und Wayne Jansen (2014). *Guidelines on Mobile Device Forensics*. 800-101 Revision 1. National Institute of Standards und Technology. DOI: 10.6028/NIST.SP.800-101r1.

Balakrishnan, Hari u. a. (2003). „Looking Up Data in P2P Systems“. In: *Communications of the ACM* 46.2, S. 43–48. DOI: https://doi.org/10.1145/606272.606299.

Bicakci, Kemal und Bulent Tavli (2009). „Denial-of-Service attacks and countermeasures in IEEE 802.11 wireless networks“. In: *Computer Standards & Interfaces* 31.5. Specification, Standards and Information Management for Distributed Systems, S. 931–941. ISSN: 0920-5489. DOI: https://doi.org/10.1016/j.csi.2008.09.038. URL: https://www.sciencedirect.com/science/article/pii/S0920548908001438.

Brünnler, Kai (2018). *Blockchain kurz & gut*. 1. Auflage. O´Reillys Taschenbibliothek. dpunkt.verlag. ISBN: 978-3-96009-070-0.

Cotton, Michelle u. a. (Aug. 2011). *Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry*. RFC 6335. DOI: 10.17487/RFC6335. URL: https://www.rfc-editor.org/info/rfc6335.

Day, Mark u. a. (Feb. 2000). *Instant Messaging / Presence Protocol Requirements*. RFC 2779. DOI: 10.17487/RFC2779. URL: https://www.rfc-editor.org/info/rfc2779.

Diffie, Whitfield und Martin Hellman (Nov. 1976). „New Directions in Cryptography". In: *IEEE Transactions on Information Theory* 22.6, S. 644–654. DOI: 10.1109/TIT.1976.1055638.

Douceur, John R. (2002). „The Sybil Attack". In: *Peer-to-Peer Systems*. Hrsg. von Peter Druschel, Frans Kaashoek und Antony Rowstron. Springer Berlin Heidelberg, S. 251–260. ISBN: 978-3-540-45748-0.

Eddy, Wesley (Aug. 2022). *Transmission Control Protocol (TCP)*. RFC 9293. DOI: 10.17487/RFC9293. URL: https://www.rfc-editor.org/info/rfc9293.

Fill, Hans-Georg und Andreas Meier (2020). *Blockchain. Grundlagen, Anwendungsszenarien und Nutzungspotenziale*. 1. Auflage. Springer Vieweg. DOI: https://doi.org/10.1007/978-3-658-28006-2.

Ford, Bryan, Dan Kegel und Pyda Srisuresh (März 2008). *State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs)*. RFC 5128. DOI: 10.17487/RFC5128. URL: https://www.rfc-editor.org/info/rfc5128.

Galuba, Wojciech und Sarunas Girdzijauskas (2009). „Peer to Peer Overlay Networks: Structure, Routing and Maintenance". In: *Encyclopedia of Database Systems*. Hrsg. von Ling Liu und M. Tamer Özsu. Boston, MA: Springer US, S. 2056–2061. ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_1215.

Hanson, M. David (1999). *Server Management*. Hrsg. von Gilbert Held. Auerbach. Kap. The Client/Server Architecture, S. 3–14. ISBN: 978-0849398230.

Hellmann, Roland (2023). *IT-Sicherheit. Methoden und Schutzmassnahmen für sichere Cybersysteme*. 2., aktualisierte und erweiterte Auflage. De Gruyter Oldenbourg.

Holdrege, Matt und Pyda Srisuresh (Aug. 1999). *IP Network Address Translator (NAT) Terminology and Considerations*. RFC 2663. DOI: 10.17487/RFC2663. URL: https://www.rfc-editor.org/info/rfc2663.

Josefsson, Simon und Ilari Liusvaara (Jan. 2017). *Edwards-Curve Digital Signature Algorithm (EdDSA)*. RFC 8032. DOI: 10.17487/RFC8032. URL: https://www.rfc-editor.org/info/rfc8032.

Kapengut, Elie und Bruce Mizrach (2023). „An Event Study of the Ethereum Transition to Proof-of-Stake“. In: *Commodities* 2.2, S. 96–110. DOI: https://doi.org/10.3390/commodities2020006.

Keränen, Ari, Christer Holmberg und Jonathan Rosenberg (Juli 2018). *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal*. RFC 8445. DOI: 10.17487/RFC8445. URL: https://www.rfc-editor.org/info/rfc8445.

Khatibi, Elahe und Mohsen Sharifi (Nov. 2020). „Resource discovery mechanisms in pure unstructured peer-to-peer systems: a comprehensive survey“. In: *Peer-to-Peer Networking and Applications* 14.2, S. 729–746.

Kunzmann, Gerald (2009). „Performance analysis and optimized operation of structured overlay networks“. In: URL: https://api.semanticscholar.org/CorpusID:4156808.

Landerreche, Esteban und Marc Stevens (2019). *On Immutability of Blockchains*. DOI: 10.18420/blockchain2018_04. URL: https://marc-stevens.nl/research/papers/ERCIM18-LS.pdf.

Langley, Adam, Mike Hamburg und Sean Turner (Jan. 2016). *Elliptic Curves for Security*. RFC 7748. DOI: 10.17487/RFC7748. URL: https://www.rfc-editor.org/info/rfc7748.

Lua, Eng Keong u. a. (2005). „A Survey and Comparison of Peer-to-Peer Overlay Network Schemes“. In: *IEEE Communications Surveys & Tutorials* 7.2, S. 72–93. DOI: 10.1109/COMST.2005.1610546.

Luntovskyy, Andriy und Dietbert Gütter (2020). *Moderne Rechnernetze. Protokolle, Standards und Apps in kombinierten drahtgebundenen, mobilen und drahtlosen Netzwerken*. 1. Auflage. Springer Vieweg Wiesbaden. ISBN: 978-3-658-25617-3. DOI: https://doi.org/10.1007/978-3-658-25617-3.

Mahlmann, Peter und Christian Schindelhauer (2007). *Peer-to-Peer-Netzwerke*. 1. Auflage. Springer-Verlag Berlin. ISBN: 978-3-540-33991-5.

Maymounkov, Petar und David Mazières (2002). *Kademlia: A Peer-to-peer Information System Based on the XOR Metric*. URL: https://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf.

Medrano-Chávez, Adan G., Elizabeth Perez-Cortès und Miguel Lopez-Guerrero (Sep. 2015). „A performance comparison of Chord and Kademlia DHTs in high churn scenarios". In: *Peer-to-Peer Networking and Applications* 8.5, S. 807–821. DOI: 10.1007/s12083-014-0294-y.

Meinel, Christoph und Tatiana Gayvoronskaya (2020). *Blockchain. Hype oder Innovation*. 1. Auflage. Springer Vieweg. DOI: https://doi.org/10.1007/978-3-662-61916-2.

Nakamoto, Satoshi (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. URL: https://bitcoin.org/bitcoin.pdf.

Perez, Daniel und Benjamin Livshits (2019). *Smart Contract Vulnerabilities: Does Anyone Care?* URL: https://allquantor.at/blockchainbib/pdf/perez2019smart.pdf.

Peris, Antonio Delgado, José M. Hernández und Eduardo Huedo (2015). „Evaluation of alternatives for the broadcast operation in Kademlia under churn". In: *Peer-to-Peer Networking and Applications* 9.2, S. 313–327. DOI: https://doi.org/10.1007/s12083-015-0338-y.

Petit-Huguenin, Marc u. a. (Feb. 2020). *Session Traversal Utilities for NAT (STUN)*. RFC 8489. DOI: 10.17487/RFC8489. URL: https://www.rfc-editor.org/info/rfc8489.

Postel, Jon (Aug. 1980). *User Datagram Protocol*. RFC 768. DOI: 10.17487/RFC0768. URL: https://www.rfc-editor.org/info/rfc768.

Prêtre, Baptiste (2005). *Attacks on Peer-to-Peer Networks*. Techn. Ber. Eidgenössische Technische Hochschule Zürich.

Reddy, Tirumaleswar u. a. (Feb. 2020). *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)*. RFC 8656. DOI: 10.17487/RFC8656. URL: https://www.rfc-editor.org/info/rfc8656.

Rosenfeld, Meni (2014). *Analysis of hashrate-based double-spending*. URL: https://arxiv.org/pdf/1402.2009.pdf.

Rowstron, Antony und Peter Druschel (2001). „Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems“. In: *Middleware 2001*. Hrsg. von Rachid Guerraoui, S. 329–350. DOI: https://doi.org/10.1007/3-540-45518-3_18.

Saint-Andre, Peter (März 2011). *Extensible Messaging and Presence Protocol (XMPP): Core*. RFC 6120. DOI: 10.17487/RFC6120. URL: https://www.rfc-editor.org/info/rfc6120.

Saroiu, Stefan, Krishna P. Gummadi und Steven D. Gribble (Aug. 2003). „Measuring and analyzing the characteristics of Napster and Gnutella hosts“. In: *Multimedia Systems* 49, S. 170–184. DOI: 10.1007/s00530-003-0088-1.

Shrestha, Rakesh und Seung Yeob Nam (2019). „Regional Blockchain for Vehicular Networks to Prevent 51% Attacks“. In: *IEEE Access* 7, S. 95033–95045. DOI: 10.1109/ACCESS.2019.2928753.

Sommerville, Ian (2018). *Software Engineering*. 10., aktualisierte Auflage. Pearson Deutschland. ISBN: 978-3-86894-344-3.

Sorge, Christoph und Artus Krohn-Grimberghe (2013). „Bitcoin - das Zahlungsmittel der Zukunft?“ In: *Wirtschaftsdienst* 93.10, S. 720–732. DOI: https://doi.org/10.1007/s10273-013-1589-y.

Stoica, Ion u. a. (2001). *Chord: A scalable peer-to-peer lookup service for Internet applications*. URL: https://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf.

Vanerio, Juan Martino und Pedro Casas (2018). *WhatsApp Calling: a Revised Analysis on WhatsApp's Architecture and Calling Service*. URL: https://dl.acm.org/doi/abs/10.1145/2940116.2940132.

Vu, Quang Hieu, Mihai Lupu und Beng Chin Ooi (2010). *Peer-to-Peer Computing*. Springer Berlin Heidelberg. DOI: https://doi.org/10.1007/978-3-642-03514-2_3.

Wang, Zhen u. a. (2018). „Enhanced Instant Message Security and Privacy Protection Scheme for Mobile Social Network Systems". In: *IEEE Access* 6, S. 13706–13715. DOI: 10.1109/ACCESS.2018.2813432.

Wong, David (2023). *Kryptografie in der Praxis. Eine Einführung in die bewährten Tools, Frameworks und Protokolle*. 1. Auflage. dpunkt.verlag. ISBN: 978-3-86490-939-9.

Yergeau, François (Nov. 2003). *UTF-8, a transformation format of ISO 10646*. RFC 3629. DOI: 10.17487/RFC3629. URL: https://www.rfc-editor.org/info/rfc3629.

Zhang, Rong und Wai Kin (Victor) Chan (Juli 2020). „Evaluation of Energy Consumption in Block-Chains with Proof of Work and Proof of Stake". In: *Journal of Physics: Conference Series* 1584.1, S. 012023. DOI: 10.1088/1742-6596/1584/1/012023. URL: https://dx.doi.org/10.1088/1742-6596/1584/1/012023.

Zhang, Shijie und Jong-Hyouk Lee (2019). „Double-Spending With a Sybil Attack in the Bitcoin Decentralized Network". In: *IEEE Transactions on Industrial Informatics* 15.10, S. 5715–5722. DOI: 10.1109/TII.2019.2921566.

Webseiten

Asymmetrische Kryptografie (Verschlüsselung) (o. D.). URL: https://www.elektronik-kompendium.de/sites/net/1910111.htm (besucht am 10. 01. 2024).

Bitcoin Energy Consumption Index (2024). URL: https://digiconomist.net/bitcoin-energy-consumption (besucht am 07. 01. 2024).

Briar: How it works (o. D.). URL: https://briarproject.org/how-it-works/ (besucht am 08. 01. 2024).

Fragen und Antworten (o. D.). URL: https://telegram.org/faq#f-kann-ich-den-server-quelltext-bekommen (besucht am 08. 01. 2024).

Greenwald, Glenn (6. Juni 2013). *NSA collecting phone records of millions of Verizon customers daily.* URL: https://www.theguardian.com/world/2013/jun/06/nsa-phone-records-verizon-court-order (besucht am 07. 01. 2024).

Nier, Hedda (2017). *Wie sich die digitale Kommunikation verändert hat.* URL: https://de.statista.com/infografik/11426/wie-sich-die-digitale-kommunikation-veraendert-hat/ (besucht am 30. 10. 2023).

Plett, Sandra (o. D.). *Was ist Instant Messaging?* URL: https://www.placetel.de/ratgeber/instant-messaging (besucht am 08. 01. 2024).

Pratap, Zubin (31. Aug. 2022). *Reentrancy Attacks and The DAO Hack.* URL: https://blog.chain.link/reentrancy-attacks-and-the-dao-hack/ (besucht am 07. 01. 2024).

Price, Rob (17. Juni 2016). *Digital currency Ethereum is cratering because of a $50 million hack.* URL: https://web.archive.org/web/20170611195628/http://uk.

businessinsider.com/dao-hacked-ethereum-crashing-in-value-tens-of-millions-allegedly-stolen-2016-6 (besucht am 07. 01. 2024).

Protokoll (o. D.). URL: https://www.novalnet.de/ecommerce-lexikon/protokoll#:~:text=Ein%20Protokoll%20ist%20eine%20Aufzeichnung,Datenaustausch%20zwischen%20zwei%20Computern%20regeln. (besucht am 08. 01. 2024).

Researcher FAQs (o. D.). URL: https://ethereumclassic.org/faqs/researchers#what-is-the-difference-between-ethereum-classic-and-ethereum (besucht am 07. 01. 2024).

Signal Website (o. D.). URL: https://www.signal.org/de/#signal (besucht am 08. 01. 2024).

signalapp (o. D.). *Overview*. URL: https://github.com/signalapp/libsignal/blob/main/README.md (besucht am 08. 01. 2024).

Statista (2023). *Beliebteste Messenger in Deutschland im Jahr 2023 [Graph]*. URL: https://de.statista.com/prognosen/999735/deutschland-beliebteste-messenger (besucht am 08. 01. 2024).

Symmetrische Kryptografie (Verschlüsselung) (o. D.). URL: https://www.elektronik-kompendium.de/sites/net/1910101.htm (besucht am 10. 01. 2024).

The Double Ratchet Algorithm (20. Nov. 2016). URL: https://signal.org/docs/specifications/doubleratchet/ (besucht am 10. 01. 2024).

Tox - The Tox Project (o. D.). URL: https://tox.chat/about.html (besucht am 08. 01. 2024).

Tox User FAQ (o. D.). URL: https://tox.chat/faq.html#how-tox-privacy (besucht am 08. 01. 2024).

Was sind digitale Signaturen? (o. D.). URL: https://www.docusign.com/de-de/wie-es-funktioniert/elektronische-signatur/digitale-signatur/digitale-signatur-faq (besucht am 10. 01. 2024).

WhatsApp Funktionen (o. D.). URL: https://www.whatsapp.com/expressyourself (besucht am 02. 01. 2024).

Where are the servers located? (o. D.). URL: https://threema.ch/en/faq/server_location#:~:text=Our%20servers%20assume%20the%20role,cannot%20be%20any%20asynchronous%20communication. (besucht am 08. 01. 2024).

XMPP Instant Messaging (o. D.). URL: https://xmpp.org/uses/instant-messaging/ (besucht am 08. 01. 2024).

XMPP Specifications (o. D.). URL: https://xmpp.org/extensions/ (besucht am 08. 01. 2024).

Zakharyan, Olya (o. D.). *Writing software requirements for your messaging service*. URL: https://yalantis.com/blog/writing-software-requirements-for-your-messaging-service/ (besucht am 07. 01. 2024).

zetok (o. D.). *Tox - The Tox Reference*. URL: https://zetok.github.io/tox-spec/ (besucht am 08. 01. 2024).