Institut für Informationssicherheit

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Entwurf und Analyse eines sicheren und effizienten Bulletin Board auf Basis von Blockchains

Tim Würtele

Studiengang:	Informatik
Prüfer/in:	Univ. Prof. Dr. Ralf Küsters
Betreuer/in:	Dipl.-Math. Mike Simon
Beginn am:	06. Dezember 2017
Beendet am:	06. Juni 2018

Zusammenfassung

In vielen durch die Wähler und unabhängige Beobachter verifizierbaren eVoting-Systemen wird ein sogenanntes Bulletin Board benötigt. Ein Bulletin Board ist eine Art öffentliches schwarzes Brett, auf dem jeder Nachrichten veröffentlichen und von dem jeder die veröffentlichten Nachrichten lesen kann. Ein solches Bulletin Board sollte mindestens die Eigenschaft haben, dass Nachrichten, die einmal veröffentlicht wurden, nicht wieder entfernt oder verändert werden können. Bisher wurden Bulletin Boards – wenn die Implementierung überhaupt berücksichtigt wurde – in der Regel entweder als zentraler Server, dem vertraut werden muss, oder als verteiltes System implementiert.

Mit den seit 2008 bekannten Blockchains gibt es eine Technologie, die ein ähnliches Versprechen gibt: Dass nur Daten angehängt, bestehende aber nicht verändert werden können.

In dieser Arbeit werden zunächst Blockchains und Bulletin Boards zusammen mit ihren wesentlichen Eigenschaften diskutiert und anschließend wird untersucht, wie sich ein Bulletin Board auf Basis bzw. mithilfe einer Blockchain implementieren lässt.

Inhaltsverzeichnis

Danksagungen 1

1 Einleitung **3**

2 Grundlagen **7**
2.1 Kryptographische Hashfunktionen 7
2.2 Verschlüsselungsverfahren 8
2.3 Public-Key-Infrastrukturen 8
2.4 Digitale Signaturen 9
2.4.1 Thresholdsignaturen 10
2.5 eVoting 11
2.6 Bemerkungen zu verteilten Systemen 12
2.6.1 Interaktionsmodelle 12
2.6.2 CAP-Theorem 13
2.6.3 Byzantine Agreement/Consensus 13

3 Öffentliche Blockchains **17**
3.1 Transaktionen 18
3.2 Blöcke 19
3.3 Mining 21
3.4 Proof of Work 24
3.5 Netzwerkstruktur 25
3.6 Eine konkrete Implementierung: Bitcoin 26
3.6.1 Transaktionen in Bitcoin 27
3.6.2 Blöcke und Mining in Bitcoin 28
3.7 Limitierungen und Probleme von Bitcoin und Blockchains allgemein 29
3.7.1 Transaktionsrate 30
3.7.2 Timestamps bei Proof of Work-Blockchains 30
3.7.3 ASIC-Mining 31
3.7.4 Energieverbrauch durch Mining bei Proof of Work-Blockchains 31
3.7.5 Blockchain Trilemma 32
3.7.6 Eclipse Attacks 32
3.8 Eigenschaften von Blockchains 33

4 Bulletin Boards **35**
4.1 Anforderungen an Bulletin Boards 35
4.1.1 Häufige Anforderungen 36

4.1.2 Veröffentlichungen speziell zu Bulletin Boards . . . 37
4.1.3 Benutzer von Bulletin Boards . . . 41
4.1.4 Übersicht und Einordnung der Anforderungen . . . 44
4.1.5 Zusammenfassung . . . 49
4.2 Angriffsszenarien und Gegenmaßnahmen . . . 50
4.2.1 Angriffe von außen auf das Bulletin Board . . . 50
4.2.2 Angriffe durch das Bulletin Board selbst . . . 51
4.3 Andere Arbeiten zu Bulletin Boards . . . 55
4.3.1 Bulletin Board in Remotegrity/Gemeindewahl Takoma Park . . . 55
4.3.2 Bulletin Board in Helios . . . 56
4.3.3 Einschub: Notation . . . 56
4.3.4 Bulletin Board von Peters . . . 57
4.3.5 Bulletin Board von Hauser und Haenni . . . 58
4.3.6 Bulletin Board in D-DEMOS . . . 59
4.3.7 Bulletin Board von Culnane und Schneider . . . 60
4.3.8 Bulletin Board von Kuldmaa . . . 62
4.3.9 Bulletin Board von Heather und Lundin . . . 62

5 Implementierungsmodelle für Bulletin Boards auf Basis von Blockchains 65
5.1 Zentrales bzw. verteiltes System als Bulletin Board . . . 66
5.2 Nachweisbarkeit von nachträglichen Änderungen bei Blockchains . . . 67
5.3 Blockchain als Bulletin Board . . . 68
5.3.1 (Neue) Angriffsszenarien durch Verwendung einer Blockchain als Bulletin Board . . . 68
5.3.2 Diskussion zentraler Bulletin Board-Eigenschaften . . . 69
5.3.3 Fazit . . . 72
5.4 Hybride Modelle aus klassischem Bulletin Board und Blockchain . . . 73
5.4.1 Untersuchung der Modelle . . . 74
5.4.2 Fazit . . . 75
5.5 Smart Contract als Bulletin Board . . . 76
5.5.1 Einführung zu Smart Contracts in Ethereum . . . 76
5.5.2 Smart Contract als Bulletin Board . . . 77
5.5.3 Smart Contract als komplettes eVoting-System . . . 78

6 Zusammenfassung und Ausblick 81

Literatur 83

Danksagungen

An dieser Stelle möchte ich mich bei all jenen bedanken, die mich im Rahmen dieser Masterarbeit begleitet und unterstützt haben. Besonders hervorheben will ich dabei meinen Betreuer Mike Simon, der sich viel Zeit für anregende Diskussionen und Verbesserungsvorschläge genommen hat. Außerdem danken möchte ich Prof. Ralf Küsters für fachliche Hinweise und Diskussionen sowie ihm und seinen Mitarbeitern Daniel Fett, Daniel Rausch und Guido Schmitz für die Unterstützung bei der Themenfindung. Darüber hinaus danke ich meinen Kommilitonen Andreas Preikschat und Pedram Hosseyni für ihre moralische Unterstützung und helfenden Hinweise, wenn ich „den Wald vor lauter Bäumen nicht gesehen habe“ und Dr. Tomasz Truderung für eine angeregte Diskussion über den praktischen Einsatz von Bulletin Boards. Zu guter Letzt will ich noch meiner Frau und meinen Eltern dafür danken, dass sie mir zum einen das Studium an sich ermöglicht und zum anderen speziell während der Arbeit an dieser Masterarbeit den Rücken frei gehalten haben.

1 Einleitung

Mit der fortschreitenden Digitalisierung aller Lebensbereiche stellt sich unter anderem auch die Frage, ob Wahlen aller Art nicht auch durch Computer unterstützt durchgeführt werden können. Davon verspricht man sich zunächst eine einfachere – weil automatische – Auszählung, die wiederum auch komplexere Abstimmungsverfahren auf der einen und häufigere Abstimmungen auf der anderen Seite ermöglichen würde, da der notwendige Aufwand pro Wahlgang sinkt. Das inhärente Problem damit ist, dass die internen Abläufe in einem Computer erstens nur sehr schwer nachvollziehbar bzw. überprüfbar sind und dass eine geschickte Manipulation der Wahlcomputer das Ergebnis signifikant beeinflussen kann, ohne dass ein Betrug nachweisbar wäre. Die bisherige Geschichte mit Wahlcomputern zeigt, dass solche Manipulationen durchaus (und zum Teil erschreckend einfach) möglich sind ([39, 49, 71, 104, 105] und viele weitere).

Darum wird schon seit Längerem an *überprüfbaren* eVoting-Systemen geforscht, bei denen bestimmte – oder alle – Schritte von Wählern und/oder externen Beobachtern verifiziert werden können. Dazu gehört z. B., dass ein Wähler überprüfen kann, ob seine Stimme korrekt im Endergebnis mitgezählt wurde; ein weiteres Beispiel ist die Überprüfbarkeit der Auszählung, d. h., dass externe Beobachter „nachzählen“ und ihr Ergebnis mit dem offiziellen vergleichen können. Sehr viele dieser überprüfbaren eVoting-Systeme benötigen jedoch ein sogenanntes *Bulletin Board* – eine Art öffentliches schwarzes Brett, auf dem jeder (Beteiligte) Nachrichten veröffentlichen kann und von dem jeder Wähler oder Wahlbeobachter alle Nachrichten lesen kann. Exemplarisch seien hier die eVoting-Systeme D-DEMOS von Chondros et al. [25], ein System von Kiayias et al. [65], sElect von Küsters et al. [72] und Helios von Ben Adida [2] genannt; es gibt aber noch viele weitere [4, 5, 8, 9, 21, 24, 29, 62, 87, 96, 109, u. v. m.].

Für ein solches Bulletin Board muss also gelten, dass einmal veröffentlichte Nachrichten nicht mehr geändert oder gelöscht werden können. Das schließt auch die mit den Nachrichten veröffentlichten Metadaten wie z. B. Zeitpunkt der Veröffentlichung, Autor und so weiter mit ein. Diese Eigenschaft macht im Kern ein Bulletin Board aus, je nach Anwendungsgebiet ist jedoch eine Vielzahl weiterer Eigenschaften gewünscht oder notwendig. Fast immer wird z. B. gefordert, dass eine (per Definition unerwünschte) Änderung der veröffentlichten Daten leicht und zuverlässig erkannt werden kann – wer das erkennt und was die Folgen einer solchen Erkennung sind, hängt dann wieder von der konkreten Anwendung ab.

Sinnvolle Anwendungen für Bulletin Boards ergeben sich immer dann, wenn die oben genannte „Kerneigenschaft“ – dass Daten nicht verändert, sondern nur ergänzt werden können – benötigt wird. Beispiele aus der Literatur umfassen neben den als Motivation eingeführten eVoting-Systemen z. B. auch Auktionen [55], öffentliche Diskussionsplattformen sowie Logs von wichtigen Systemen [103].

Insbesondere an der Anwendung in Wahlsystemen wird jedoch bereits deutlich, dass ein Bulletin Board nicht für alle Anwendungen sinnvoll zentralisiert betrieben werden kann: Der Betreiber könnte seine zentrale Position ausnutzen, um die veröffentlichten Daten im Sinne seiner Interessen zu manipulieren bzw. um bestimmte Daten gar nicht erst zu veröffentlichen. Gerade letzteres ist in der Regel nicht zweifelsfrei nachweisbar. Ein Ausweg ist daher, das Bulletin Board als verteiltes System auszuführen, bei dem sich die einzelnen – auf möglichst viele Interessensgruppen verteilten – Teile des Bulletin Boards sowohl gegenseitig kontrollieren als auch von außen überprüft werden können. Beispiele für diesen Ansatz sind die Arbeiten von Chondros et al. [25] sowie Culnane und Schneider [30], die später noch im Detail betrachtet werden.

Wie bereits erwähnt, muss ein Bulletin Board in der Regel noch weitere Anforderungen erfüllen, um praktisch eingesetzt werden zu können. Beispiele sind eine eindeutige Reihenfolge der veröffentlichten Daten [55], ein definierter Endzeitpunkt, ab dem keine neuen Daten mehr angenommen werden [30] oder die ausschließliche Annahme authentifizierter Daten [1, 55]. Dazu kommen noch Anforderungen an die *Lebendigkeit*[1] im weitesten Sinne, z. B. eine gewisse Anzahl an Nachrichten, die pro Sekunde verarbeitet werden müssen.

Eine Technologie, die möglicherweise die Entwicklung zukünftiger Bulletin Boards erleichtern könnte, ist die Blockchain. Eine Blockchain ist ein dezentrales „Transaktionsverzeichnis“, sie speichert also Transaktionen und deren Reihenfolge. Dabei werden die einzelnen Transaktionen in sogenannten Blöcken gruppiert und diese Blöcke wiederum durch Verweise auf den jeweiligen Vorgängerblock verknüpft – so entsteht eine „Kette von Blöcken“: eine „Blockchain“. Identifiziert wird ein Block dabei durch einen Hashwert – also eine Prüfsumme – über seinen Inhalt. Das führt dazu, dass der Inhalt eines Blocks nicht verändert werden kann, ohne dass auch die nachfolgenden Blöcke geändert werden müssen: Durch die initiale Änderung ändert sich der Hashwert des initial geänderten Blocks, dieser Wert ist aber Teil des nächsten Blocks, folglich muss dieser angepasst werden, wodurch sich wieder dessen Hashwert ändert und so weiter.

Nun kann es vorkommen, dass sich die Kette aufspaltet, weil zwei unterschiedliche Blöcke auf den gleichen Vorgänger verweisen. Damit sich die an der Blockchain Beteiligten dann „einig“ sind, welche Kette nun die gültige ist, wird immer die längste[2] Kette aus gültigen Blöcken als aktuell gültige Kette angesehen. Weil außerdem das Erstellen gültiger Blöcke limitiert wird – z. B. weil dabei Kosten entstehen – ist es nicht ohne Weiteres möglich, an einem bestehenden Block der gültigen Kette etwas zu ändern: Wie oben beschrieben, würde sich eine solche Änderung nicht nur auf den manipulierten, sondern auch auf alle nachfolgenden Blöcke auswirken. Folglich müssten alle diese Blöcke neu erstellt werden – was wegen der angesprochenen Limitierung in der Praxis quasi unbezahlbar ist.

Das bedeutet, dass eine Blockchain auf den ersten Blick die wichtigste Anforderung an ein Bulletin Board erfüllt: Es können nur Daten hinzugefügt werden (in Form von neuen Blöcken), aber keine zu einem früheren Zeitpunkt eingefügten Daten verändert oder entfernt werden.

[1] Unter „Lebendigkeit“ werden jene Eigenschaften eines Systems zusammengefasst, die Aussagen über die Verfügbarkeit und die verfügbare Qualität und Performance der verschiedenen Funktionen des Systems bei bestimmten Umgebungsbedingungen machen.

[2] Besser und später im Detail erklärt: „schwerste Kette“ – hier zur Vereinfachung aber zunächst „längste Kette“.

Zudem sind (öffentliche) Blockchains von Anfang an dezentral angelegt und setzen keinerlei Vertrauen der einzelnen Teilnehmer untereinander voraus – was ebenfalls gut zu einem Bulletin Board passen würde.

Diese Masterarbeit beschäftigt sich daher mit der Frage, inwiefern sich Eigenschaften einer Blockchain nutzen lassen, um ein möglichst sicheres und effizientes Bulletin Board zu konstruieren. Nach einigen Grundlagen in Kapitel 2 folgt in Kapitel 3 eine Einführung in die wichtigsten Aspekte und Eigenschaften von öffentlichen Blockchains. Bulletin Boards und die von ihnen geforderten Eigenschaften werden in Kapitel 4 ausführlich diskutiert. In Kapitel 5 folgt dann eine Untersuchung verschiedener Möglichkeiten, ein Bulletin Board auf Basis oder mithilfe einer Blockchain zu implementieren; Kapitel 6 fasst die Arbeit und wichtigsten Ergebnisse zusammen und gibt einen Ausblick für zukünftige Arbeiten.

2 Grundlagen

In diesem Kapitel werden die benötigten Grundlagen aus der Kryptographie sowie einige Aspekte verteilter Systeme präsentiert. Die Abschnitte sind dabei nicht als ausführliche Einführung in das jeweilige Thema zu verstehen, sondern sollen nur so viel Information vermitteln, wie zum Verständnis der nachfolgenden Kapitel notwendig ist.

2.1 Kryptographische Hashfunktionen

Eine Hashfunktion[1] ist eine Funktion $h : \{0,1\}^* \to \{0,1\}^n$, d. h., sie bildet Eingaben beliebiger Länge auf Ausgaben fester Länge ab. Das bedeutet insbesondere, dass eine Hashfunktion nicht injektiv sein kann – tatsächlich gibt es mindestens ein Element in der Bildmenge, das unendlich viele Urbilder hat.

Weil Hashfunktionen später als kryptographische Prüfsummen verwendet werden sollen, sollten sie trotzdem „so injektiv wie möglich“ und möglichst schwer vorhersagbar sein. An *kryptographische* Hashfunktionen werden deshalb zusätzliche Anforderungen gestellt: Sie müssen *kollisionsresistent* sein, d. h., es soll praktisch unmöglich ist, zwei verschiedene Eingaben x, y zu finden, sodass $h(x) = h(y)$ ist. Das impliziert unter anderem, dass es praktisch unmöglich ist, zu einem gegebenen Hashwert y eine Eingabe x zu finden, sodass $h(x) = y$ ist. Und daraus wiederum folgt, dass es ebenso praktisch unmöglich ist, aus einem Hashwert brauchbare Rückschlüsse auf die Eingabe zu ziehen.

Eingesetzt werden Hashfunktionen, wie bereits erwähnt, häufig als Prüfsummen: Um sicherzustellen, dass ein Dokument auf dem Transportweg nicht verändert wurde, muss nicht das ganze Dokument auf sicherem Wege transportiert bzw. gespeichert werden, es genügt, den Hashwert des Dokuments sicher zu transportieren. Zusätzlich muss dieser Hashwert auch für vertrauliche Dokumente nicht geheim gehalten werden, da aus dem Hashwert nicht auf das Dokument geschlossen werden kann. Das wiederum ermöglicht eine weitere Anwendung von Hashfunktionen: Sie können verwendet werden, um zu beweisen, dass ein bestimmtes Dokument zu einem früheren Zeitpunkt bereits existierte – sobald das Dokument fertig ist, wird ein Hashwert über dieses Dokument gebildet und in geeigneter Weise mit einer Zeitangabe veröffentlicht. Sollte es dann später Zweifel daran geben, ob das Dokument zum behaupteten

[1] Die hier verwendete Definition ist informell, für die Zwecke dieser Arbeit aber ausreichend. Für eine genauere Definition und Diskussion wird auf die einschlägige Literatur verwiesen, z. B. die entsprechenden Kapitel in [14, 36] und [97].

Zeitpunkt existierte, kann als Beweis das Dokument veröffentlicht werden, so kann jeder den Hashwert selbst bilden und mit dem ursprünglich veröffentlichten Wert vergleichen.

Aus der Kollisionsresistenz kryptographischer Hashfunktionen folgt noch eine weitere interessante Eigenschaft, die später für Proof of Work-Algorithmen wichtig wird: Es ist sehr schwer, eine Eingabe für eine solche Hashfunktion zu finden, sodass die Ausgabe in einem bestimmten Wertebereich liegt – idealerweise bleibt einem nichts anderes übrig, als so lange zufällige Eingaben „durchzuprobieren“, bis die Ausgabe im gewünschten Wertebereich liegt.

Im Rest dieser Arbeit wird nicht weiter auf die Unterscheidung zwischen kryptographischen und „normalen“ Hashfunktionen eingegangen, mit „Hashfunktion“ ist im Folgenden also immer eine kryptographische Hashfunktion gemeint.

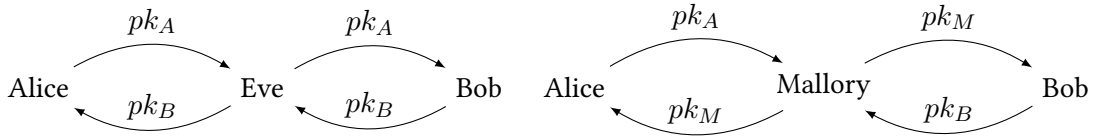
2.2 Verschlüsselungsverfahren

Wollen zwei Parteien Alice A und Bob B sich gegenseitig geheime Nachrichten über einen unsicheren Kanal senden, dann müssen sie ihre Nachrichten verschlüsseln. Prinzipiell stehen dafür zwei Klassen von Verschlüsselungsverfahren zur Verfügung: symmetrische und asymmetrische. Bei einem symmetrischen Verschlüsselungsverfahren verwenden Alice und Bob beide denselben (geheimen) Schlüssel k_{AB}, um ihre Nachrichten zu ver- und entschlüsseln. Ein offensichtliches Problem daran ist, dass sich Alice und Bob dafür zunächst auf einen Schlüssel einigen müssen – und zwar so, dass nur sie beide ihn kennen. Für dieses Problem gibt es praxistaugliche Lösungen wie z. B. den Diffie-Hellman-Schlüsselaustausch [31]. Ein weiteres subtileres Problem bei symmetrischen Verfahren ist die Anzahl der nötigen Schlüssel: Wollen Alice und Bob jeweils auch mit Charlie C kommunizieren, braucht es bereits drei verschiedene Schlüssel: k_{AB}, k_{AC}, k_{BC}, im Allgemeinen sind für n Kommunikationspartner $\mathcal{O}(n^2)$ Schlüssel nötig.

Dieses Problem lösen asymmetrische Verfahren wie z. B. RSA [94] und ElGamal [35]: In einem asymmetrischen Verfahren hat jeder Kommunikationspartner P zwei zusammengehörige Schlüssel, einen öffentlichen Schlüssel pk_P und einen geheimen Schlüssel sk_P. Der öffentliche Schlüssel pk_P darf dabei – wie der Name suggeriert – öffentlich bekannt sein; er dient dazu, Nachrichten an P zu verschlüsseln, diese Nachrichten können dann nur mithilfe des geheimen Schlüssels sk_P wieder entschlüsselt werden (nicht aber mit pk_P). Damit sind für n Kommunikationspartner nur noch n Schlüsselpaare notwendig.

2.3 Public-Key-Infrastrukturen

Für die in Abschnitt 2.2 vorgestellten Verschlüsselungsverfahren müssen Alice und Bob jeweils vor Beginn der verschlüsselten Kommunikation ihre(n) Schlüssel austauschen. Mit den asymmetrischen Verfahren ist das mit einem rein passiven Angreifer Eve kein Problem, Alice und Bob können sich einfach ihre öffentlichen Schlüssel schicken, wie in Abbildung 2.1a dargestellt.



(a) Passiver Angreifer

(b) Aktiver Angreifer als „Man-in-the-Middle“

Abbildung 2.1: Schlüsselaustausch zwischen Alice und Bob. Anschließend verwenden Alice und Bob die erhaltenen Schlüssel, um Nachrichten zu verschlüsseln.

Gibt es jedoch einen aktiven Angreifer Mallory M, dann könnte dieser den Schlüsselaustausch zwischen A und B wie in Abbildung 2.1b dargestellt angreifen und anschließend Nachrichten beliebig lesen, verändern oder eigene „erfinden“ – Alice und Bob haben zunächst keine Möglichkeit, das zu bemerken. Es sei hier auch ausdrücklich darauf hingewiesen, dass dieses Problem analog auch beim Austausch/Aushandeln eines symmetrischen Schlüssels besteht (z. B. mittels Diffie-Hellman-Schlüsselaustausch).

Das zugrunde liegende Problem ist, dass Alice und Bob jeweils nicht verifizieren können, ob der Schlüssel, den sie erhalten, wirklich dem jeweils anderen gehört. Und an dieser Stelle setzen Public-Key-Infrastrukturen an: Sie sorgen dafür, dass z. B. Alice überprüfen kann, ob ein Schlüssel pk_x, den sie erhalten hat, tatsächlich zu Bob gehört. Die Details einer solchen Public-Key-Infrastruktur gehen über den Rahmen dieser Arbeit hinaus, der interessierte Leser sei für übersichtliche Einführungen an [36, Kap. 7] und [97, Kap. 8.12] verwiesen.

2.4 Digitale Signaturen

Mit den in Abschnitt 2.2 eingeführten Verschlüsselungsverfahren können Alice und Bob sich gegenseitig Nachrichten schicken und dank einer Public-Key-Infrastruktur können sie sich dabei auch sicher sein, dass sie den richtigen Schlüssel verwenden. Aber es gibt noch ein Problem: Da die öffentlichen Schlüssel (logischerweise) öffentlich sind, kann sich jeder gegenüber Alice als Bob ausgeben. Außerdem gibt es keine Möglichkeit, gegenüber Dritten zu beweisen, wer eine Nachricht versendet hat – das ist z. B. für digitale Verträge notwendig, Alice muss ja im Zweifel gegenüber einem Gericht beweisen können, dass Bob dem Vertrag tatsächlich zugestimmt hat.

Genau das kann mithilfe von digitalen Signaturen erreicht werden. Eine Möglichkeit[2], ein solches Signaturschema zu implementieren, besteht darin, ein asymmetrisches Verschlüsselungsverfahren „umgekehrt“ zu benutzen (das Verschlüsselungsverfahren muss das natürlich unterstützen): Will Alice eine Nachricht m signieren, verschlüsselt sie den Hashwert $h(m)$ mit ihrem *privaten* Schlüssel sk_A: $\mathrm{Sig}_A(m) := \mathrm{enc}_{sk_A}(h(m))$ und verschickt dann $(\mathrm{Sig}_A(m), m)$

[2] Tatsächlich ist es etwas komplizierter, insbesondere wird in der Regel nicht einfach nur ein Hashwert ver- und wieder entschlüsselt – eine ausführliche Diskussion digitaler Signaturverfahren würde hier aber den Rahmen sprengen.

an Bob. Dieser wiederum kann nun den *öffentlichen* Schlüssel pk_A von Alice nutzen, um die Signatur zu prüfen: Er berechnet seinerseits $h(m)$, „entschlüsselt" mithilfe von pk_A den Hashwert und vergleicht die beiden Werte – sind sie identisch, ist die Signatur gültig. Diese Prüfung kann auch jeder andere durchführen und weil nur der Besitzer von sk_A – also Alice – in der Lage ist, eine gültige Signatur zu erzeugen, ist damit klar, dass die Nachricht von Alice unterschrieben wurde.

Da die Signatur außerdem durch den Hashwert an das signierte Dokument gebunden ist, ergeben sich folgende Eigenschaften von digitalen Signaturen[3]:

Fälschungssicher Die Signatur wurde wirklich vom Inhaber des geheimen Schlüssels erstellt, kein anderer konnte diese Unterschrift erstellen.

Nicht wiederverwendbar Die Signatur ist nur für das initial signierte Dokument gültig.

Unveränderbar Das signierte Dokument kann nicht geändert werden, ohne dass die Signatur ungültig wird.

Bindend Der Unterzeichner kann nicht abstreiten, dass die Signatur von ihm stammt.

Es sei an dieser Stelle darauf hingewiesen, dass eine *fälschungssichere* Signatur nicht automatisch auch *bindend* ist, letzteres erfordert nämlich, dass die Signatur auch ohne Mitwirkung des Unterzeichners überprüft und ihm zugeordnet werden kann.

2.4.1 Thresholdsignaturen

Eine spezielle Form von digitalen Signaturen sind die sogenannten Threshold- oder deutsch Schwellwertsignaturen. Dabei wird ein Geheimnis (z. B. ein geheimer Schlüssel) auf n Parteien $P_1, ..., P_n$ verteilt und zwar so, dass mindestens $k \leq n$ der Parteien ihren Teil des Geheimnisses beisteuern müssen, damit eine gültige Signatur erzeugt werden kann.

Die Details variieren je nach verwendetem Schema, im Rest der Arbeit wird davon ausgegangen, dass eine Thresholdsignatur folgendermaßen funktioniert: Es gibt eine Funktion combine, welche aus (mindestens) k Teilsignaturen $\text{SigS}_{P_i}(m)$ über eine Nachricht m eine gültige Signatur $\text{TSig}(m)$ erzeugt:

$$\text{TSig}(m) = \text{combine}\left(\text{SigS}_{P_{i_1}}(m), \text{SigS}_{P_{i_2}}(m), ..., \text{SigS}_{P_{i_k}}(m)\right) \tag{2.1}$$

Solche Schemata existieren, ein Beispiel ist das von Shoup [98].

[3] Größtenteils aus [36, S. 105] übernommen.

2.5 eVoting

Da die mit Abstand populärste Anwendung für Bulletin Boards eVoting-Systeme sind, wird hier eine kurze Einführung in die Welt des eVoting gegeben und eine grobe Abgrenzung zwischen Bulletin Board und eVoting-System vorgenommen.

An ein Wahlsystem werden – unabhängig davon, wie es genau implementiert ist – in der Regel mindestens die folgenden Anforderungen gestellt:

- Geheim: Die Stimmabgabe soll geheim sein.
- Zugangsbeschränkt: Nur Stimmabgaben von Stimmberechtigten werden gezählt.
- Gleich: Jeder Stimmberechtigte kann nur einmal abstimmen (mit der jeweils gleichen Anzahl an Stimmen).
- Fair: Es gibt keine Zwischenergebnisse während die Wahl läuft (d. h. jeder Wähler hat dieselben Informationen zum Zeitpunkt der Stimmabgabe).
- Korrekt: Das Endergebnis entspricht der Summe der abgegebenen, gültigen Stimmen.
- Verifizierbar: Die Korrektheit kann überprüft werden.
- Coercion Resistance: Kein Wähler kann zu einer bestimmten Stimmabgabe gezwungen werden bzw. seine Stimme mit Nachweis verkaufen.
- Verständlich: Ein „durchschnittlicher“ Wähler soll das System verstehen und nachvollziehen können, wie es funktioniert.

Klassische papierbasierte Wahlsysteme erfüllen diese Eigenschaften – sofern es genügend Wahlbeobachter gibt – relativ gut, haben allerdings auch einige Nachteile. Offensichtlich ist, dass das Auszählen von Hand relativ langsam und fehleranfällig ist, besonders dann, wenn das Wahlsystem komplexe Stimmabgaben erlaubt (z. B. mehrere Stimmen pro Wähler, die auf mehrere Kandidaten verteilt werden können). Außerdem kann bei größeren Wahlen mit mehreren Wahlorten kein Einzelner die ganze Wahl verifizieren, auch der einzelne Wähler kann sich nicht sicher sein, dass *seine* Stimme im Endergebnis enthalten ist.

Diese Schwächen versuchen eVoting-Systeme zu lösen, indem sie zusätzlich – unter anderem durch den Einsatz von Kryptographie – folgende Ziele verfolgen (nicht jedes eVoting-System versucht, jedes dieser Ziele zu erreichen):

- Ende-zu-Ende verifizierbar: Jeder einzelne Wähler kann nachvollziehen, dass seine Stimme korrekt im Endergebnis enthalten ist.
- Überprüfbare Auszählung: Jeder (auch Nichtwähler) kann überprüfen, dass die abgegebenen Stimmen korrekt ausgezählt wurden.
- *Accountability*: Wenn eine an der Wahl beteiligte Entität betrügt, kann ihr das *nachgewiesen* werden.

- *Remote*: Die Wähler sollen ihre Stimme von jedem beliebigen Ort aus abgeben können, üblicherweise über das Internet.

Mit Blick auf die reichhaltige Literatur[4] zu diesem Thema ist allerdings festzuhalten, dass es bisher augenscheinlich kein eVoting-System gibt, welches alle Anforderungen erfüllt[5].

Wichtig für den Rest dieser Arbeit ist noch die Abgrenzung zwischen dem eVoting-System und dem Bulletin Board bezüglich der oben aufgeführten Anforderungen an ein Wahlsystem. Hier gilt grundsätzlich, dass das Bulletin Board keine dieser Aufgaben direkt übernimmt – der Grund ist, dass diese Arbeit ein möglichst allgemeines Bulletin Board zum Ziel hat, das unter anderem auch außerhalb von eVoting-Systemen verwendet werden kann. Das bedeutet z. B., dass auf dem Bulletin Board zunächst jeder (auch Nichtwähler) Nachrichten veröffentlichen kann, das eVoting-System muss dann entsprechende Vorkehrungen (z. B. durch digitale Signaturen) treffen, um bei der Auszählung gültige von ungültigen Stimmen zu unterscheiden[6].

2.6 Bemerkungen zu verteilten Systemen

Ein verteiltes System ist ein System aus unabhängigen „Prozessen“ – hier Computern/Servern – die gemeinsam eine Aufgabe erfüllen sollen. Sowohl im Kontext von Blockchains als auch im Kontext von Bulletin Boards wird im Folgenden immer wieder auf einige Grundlagen zu verteilten Systemen zurückgegriffen. Diese Grundlagen werden hier kurz erläutert, dieser Abschnitt hat aber nicht den Anspruch, eine umfassende Einführung zu verteilten Systemen zu bieten.

2.6.1 Interaktionsmodelle

Für ein verteiltes System ist das Verhalten des Netzwerks, über das die Prozesse miteinander kommunizieren können, sowie das Zeitverhalten der einzelnen Prozesse von entscheidender Bedeutung. Beides zusammen wird dabei in einem Interaktionsmodell festgehalten.

Grundsätzlich bewegen sich die Interaktionsmodelle für verteilte Systeme zwischen zwei Extremen: Auf der einen Seite stehen die *synchronen* Systeme, auf der anderen die *asynchronen*. In einem synchronen System gibt es eine *bekannte* obere Schranke für die Laufzeit von Nachrichten im Netzwerk, für jeden Prozess und Protokollschritt sind obere und untere Schranken für die Ausführungszeit bekannt und jeder Prozess hat eine lokale Uhr mit *bekannter, beschränkter* Driftrate[7]. In einem asynchronen System dagegen gibt es keine Schranken für die Laufzeit von Nachrichten, Ausführungszeit von Prozessen und Driftrate der Uhren[8].

[4] [1, 2, 4–6, 8, 9, 12, 21, 24, 25, 27–29, 42, 62, 64–66, 72, 76, 79, 87, 96, 101, 102, 109, 110], um nur einige zu nennen.
[5] Vergleiche Küsters et al.: „[...] there does not seem to exist a 'one size fits all' remote e-voting system.“ [72, S. 1]
[6] Für eine reale Implementierung wäre auch denkbar, solche häufig benötigten Funktionen in Form von „Plugins“ zum Bulletin Board hinzuzufügen – im Rahmen dieser Arbeit wird das aber nicht weiter verfolgt.
[7] Die Driftrate einer Uhr gibt an, wie schnell sich diese Uhr von der Echtzeit entfernt.
[8] Es gibt auch noch andere Definitionen für (a)synchrone Systeme, für synchrone Systeme z. B. gibt es auch eine rundenbasierte Definition – im Ergebnis sind die damit erreichten Eigenschaften aber ähnlich.

Dazwischen finden sich viele Abstufungen, für die Belange dieser Arbeit braucht es neben synchronen und asynchronen noch die Notation der teilsynchronen Systeme. In einem teilsynchronen System (innerhalb dieser Arbeit) gibt es (endliche) Schranken für Nachrichtenlauf- und Ausführungszeit sowie für die Driftrate aller Uhren, diese Schranken sind mit Ausnahme der Nachrichtenlaufzeit jedoch nicht bekannt.

Das Interaktionsmodell hat außerdem direkte Auswirkungen darauf, wie gut (oder schlecht) die Uhren in einem verteilten System synchronisiert werden können: In einem synchronen System lassen sich Uhren – abhängig von den konkreten Zeit- und Driftschranken – sehr einfach synchronisieren, insbesondere ist die maximale Abweichung zweier Uhren zu jedem Zeitpunkt bekannt. In einem asynchronen System sind dagegen überhaupt keine Garantien bezüglich der Abweichung von lokalen Uhren möglich. Für ein teilsynchrones System können zwar Garantien gegeben werden, diese beinhalten allerdings unbekannte Größen – die Ausführungszeit- und Driftschranken. Das bedeutet, dass es unabhängig vom gewählten Interaktionsmodell Abweichungen zwischen den lokalen Uhren der Prozesse geben kann – vom Interaktionsmodell hängt lediglich ab, ob der Betrag der Abweichung abgeschätzt werden kann oder nicht.

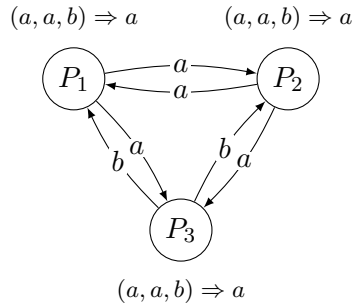
2.6.2 CAP-Theorem

Das CAP-Theorem [48] besagt, dass in einem verteilten System zu jedem Zeitpunkt nur zwei der drei Eigenschaften *Consistency*, *Availability* und *Partition Tolerance* erreicht werden können. Dabei bedeutet *Consistency*, dass alle Prozesse genau die gleichen Daten vorhalten, *Availability* steht dafür, dass das System seine Aufgabe vollumfänglich erfüllen kann und *Partition Tolerance* bezeichnet die Fähigkeit des Systems, mit dem Ausfall von Kommunikationsverbindungen umzugehen. Auf Partition Tolerance kann dabei nicht verzichtet werden: Wenn es eine Partitionierung gibt, reagiert das System darauf immer irgendwie – insbesondere aber hat das verteilte System selbst überhaupt keinen Einfluss darauf, ob es zu einer Netzwerkpartitionierung kommt oder nicht.

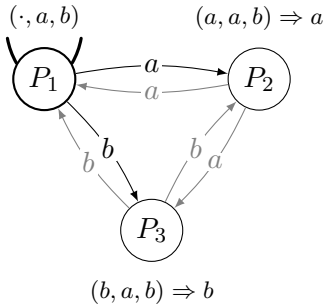
Die Aussage kann intuitiv an einer einfachen Datenbank aus zwei Prozessen nachvollzogen werden: Wenn ein Benutzer in diese Datenbank schreiben möchte, dann muss dieser Schreibvorgang – um Consistency zu erhalten – auf *beiden* Prozessen zugleich ausgeführt werden. Das wiederum setzt natürlich voraus, dass es einen funktionierenden Kommunikationskanal zwischen den beiden Prozessen gibt. Fällt dieser Kommunikationskanal aus (Netzwerkpartitionierung), so hat jeder der beiden Prozesse bei einer Schreibanfrage eines Benutzers zwei Möglichkeiten: Entweder er nimmt die Anfrage an, verändert also seine lokalen Daten und nimmt eine Verletzung von Consistency in Kauf. Oder er lehnt die Anfrage ab und erhält so Consistency, verletzt aber Availability.

2.6.3 Byzantine Agreement/Consensus

Ein häufiges Problem in einem verteilten System besteht darin, dass die Prozesse sich über einen Wert – z. B. das Ergebnis einer Berechnung – einig werden müssen. Dabei gilt es im



(a) Alle Prozesse funktionieren korrekt und kommen so alle zum selben Ergebnis a. Sollte hier ein Prozess ausfallen (Crash Failure), hat dies keinen Einfluss auf die Einigung der anderen.



(b) P_1 funktioniert nicht mehr korrekt (byzantinischer Fehler) und sendet unterschiedliche Werte an P_2 und P_3, die so zu unterschiedlichen Ergebnissen kommen.

Abbildung 2.2: Drei Prozesse einigen sich durch Mehrheitsentscheid auf einen Wert – einmal sind ausschließlich Crash Failures erlaubt, einmal byzantinische Fehler.

Wesentlichen, drei Eigenschaften zu erfüllen: Erstens soll das Ergebnis konsistent sein, d. h. alle korrekten Prozesse müssen zum selben Ergebnis kommen. Zweitens soll das Protokoll terminieren, d. h. in endlicher Zeit eine Einigung unter den korrekten Prozessen herbeiführen. Und drittens muss das Protokoll korrekt sein, d. h. wenn alle korrekten Prozesse mit demselben Wert starten, müssen sie sich auch auf genau diesen Wert einigen.

Unter der Annahme, dass jeder Prozess entweder korrekt funktioniert oder gar nichts mehr tut (*Crash Failures*), ist das relativ einfach, wie Abbildung 2.2a illustriert: Die Prozesse schicken sich gegenseitig ihre „Vorschläge“ und der am häufigsten genannte „gewinnt“. Dazu braucht es nur noch eine Methode, um Pattsituationen zu lösen, z. B. „kleinster Wert“.

Lässt man diese Annahme aber fallen und erlaubt beliebige – sog. *byzantinische* – Fehler, dann reicht eine einfache Abstimmung nicht mehr, wie das Beispiel in Abbildung 2.2b zeigt: Weil P_1 verschiedene Werte an die anderen Prozesse sendet, kommen die *korrekt funktionierenden* Prozesse P_2 und P_3 zu unterschiedlichen Resultaten, d. h. das Ergebnis ist nicht mehr konsistent.

Protokolle, die solche byzantinischen Fehler erlauben und trotzdem (in jedem Fall) mit einem korrekten, konsistenten Ergebnis terminieren, nennt man *Consensus-* oder *Byzantine Agreement*-Protokolle[9]. Diese benötigen in der Regel eine zunächst erstaunlich große Anzahl an Nachrichten (oft exponentiell in der Anzahl der Prozesse) und mindestens $t + 1$ Runden, wenn t die Anzahl der fehlerhaften Prozesse ist, die das Protokoll toleriert [40]. Zudem haben Lamport et al. [74] gezeigt, dass bei n Prozessen $t < n/3$ sein muss, um Konsistenz und Terminierung erreichen zu können.

[9] Nicht zu verwechseln mit dem verwandten Problem der „Byzantinischen Generäle“ [74]: Dabei gibt es einen Anführer, der einen Wert vorgibt – Korrektheit eines Protokolls ist dann so definiert, dass bei korrektem Anführer alle korrekten Prozesse den Wert des Anführers übernehmen.

Mithilfe von Kryptographie (vor allem digitalen Signaturen) sind auch effizientere Protokolle möglich, insbesondere kann die Anzahl der Nachrichten drastisch reduziert werden.

3 Öffentliche Blockchains

Eine Blockchain ist zunächst ein „Transaktionsverzeichnis“, in dem die einzelnen Transaktionen in Blöcken gruppiert werden. Die Blöcke wiederum sind durch Hashwerte verkettet, wie in Abbildung 3.1 dargestellt: Jeder Block enthält den Hashwert über seinen Vorgänger und eine geordnete Liste an Transaktionen. Durch die Verweise auf die Vorgänger bilden die Blöcke eine zusammenhängende Kette von Blöcken – eine Blockchain. Nun kann es passieren, dass die Blöcke keine lineare Kette, sondern einen Baum bilden – nämlich dann, wenn sich zwei unterschiedliche Blöcke auf denselben Vorgänger beziehen (siehe Abbildung 3.2 für ein Beispiel). In diesem Fall gilt der längere[1] „Ast“ des Baums („*Fork*“) als der aktuelle Zustand des Transaktionsverzeichnisses.

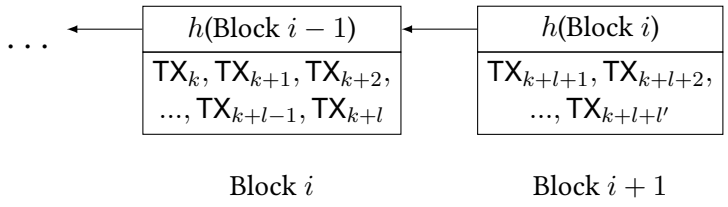


Abbildung 3.1: Einfaches Transaktionsverzeichnis, bei dem die Transaktionen TX_i in Blöcken gruppiert sind, welche ihrerseits durch eine Hashkette verkettet sind.

Das erste Mal taucht eine solche Blockchain 2008 in einem Beitrag von Satoshi Nakamoto [81] auf der Cipherpunk-Mailingliste auf[2]. In diesem Beitrag schlägt Nakamoto eine digitale Währung ohne zentrale Instanzen wie Banken vor, die er *Bitcoin* nennt. Die Blockchain selbst ist dabei eher ein „Nebenprodukt“: Wie viele andere digitale Währungen zuvor verwendet auch Bitcoin ein Transaktionsverzeichnis, in dem alle Geldbewegungen festgehalten werden. Weil Bitcoin ohne zentrale Instanzen auskommen sollte, brauchte Nakamoto ein Verfahren, mit dem sich mehrere einander misstrauende Parteien auf eine Sequenz von Transaktionen – also den Inhalt des Transaktionsverzeichnisses – einigen können. Dazu legt Nakamoto dar, wie bereits bekannte Bausteine[3] zu einer Blockchain zusammengesetzt werden können.

Etwas abstrakter betrachtet stellt eine Blockchain also ein verteiltes System dar, dessen Aufgabe

[1] Genauer: „schwerere“ Kette/Ast – der (wichtige) Unterschied wird später noch erläutert.

[2] Der Name Satoshi Nakamoto ist ein Pseudonym, seine wahre Identität ist der Öffentlichkeit bis heute unbekannt.

[3] Die Idee, Daten in Blöcken zu gruppieren und diese mithilfe von Hashwerten zu verketten, taucht z. B. fast zwanzig Jahre früher bei einem Zeitstempel-Service von Haber und Stornetta [51] auf. Auch der später noch diskutierte Proof of Work wurde schon 1992 von Dwork und Naor [34] zur Spambekämpfung vorgeschlagen.

es ist, ein Transaktionsregister zu führen, bei dem sich die einzelnen Parteien[4] aber gegenseitig nicht vertrauen – aus der Perspektive der verteilten Systeme also byzantinische Fehler aufweisen können.

Damit sich die Parteien auf eine Abfolge der Transaktionen einigen können, braucht es einen *Konsens-Algorithmus*. Dieser soll unter anderem sicherstellen, dass nicht einfach eine Partei beliebig viele gültige Blöcke erzeugen und so den Zustand des Transaktionsverzeichnisses manipulieren kann – weil ja die längste Kette an Blöcken den aktuellen Zustand darstellt. Bei Nakamotos Blockchain wird dafür ein sogenannter Proof of Work verwendet: Eine Art kryptographisches Rätsel, das nur durch den Einsatz von viel Rechenleistung gelöst werden kann. Wer dabei als Erstes die Lösung dieses Rätsels findet, darf den nächsten Block erstellen – d. h. offene Transaktionen auswählen[5] und zu einem Block bündeln – und erhält dafür eine Belohnung (dazu später mehr).

Wie die Überschrift schon andeutet, werden hier nur *öffentliche* Blockchains behandelt, d. h. Blockchains, an denen jeder teilnehmen kann. Neben den öffentlichen Blockchains gibt es noch *permissioned* Blockchains, die jeweils nur von einem begrenzten Teilnehmerkreis verwendet werden können. Bei einer permissioned Blockchain gibt es eine Instanz, die regelt, wer an der Blockchain teilnehmen darf – wer Transaktionen einbringen, wer Blöcke erstellen und ggf. auch, wer die Blöcke und Transaktionen überhaupt sehen darf. Weil diese Instanz in der Regel zentral ist, sind permissioned Blockchains meistens nicht mehr ganz dezentral. Da bei einer permissioned Blockchain die beteiligten Parteien in der Regel ein direktes Interesse an der „Gesundheit“ der Blockchain haben, braucht es bei diesen nicht unbedingt eine digitale Währung, um Miner zur Teilnahme zu motivieren. Von der „Zugangskontrolle“ und der manchmal „fehlenden“ digitalen Währung abgesehen, sind permissioned Blockchains den hier behandelten öffentlichen Blockchains in der Regel sehr ähnlich.

Der Rest dieses Kapitels ist wie folgt aufgebaut: Zunächst werden abstrakt die einzelnen Bestandteile einer Blockchain erläutert, danach folgt mit der Beschreibung von Bitcoin ein Einblick in eine konkrete Implementierung. Diese wird gefolgt von einer Diskussion einiger Limitierungen und Nachteile von Blockchains mit einem besonderen Augenmerk auf Bitcoin. Zum Schluss werden die wichtigsten Eigenschaften einer Blockchain zusammengefasst.

3.1 Transaktionen

Wie in der Einführung bereits erwähnt, werden in einer Blockchain *Transaktionen* aufgezeichnet. Das bedeutet, dass z. B. bei einer digitalen Währung wie Bitcoin keine „Kontostände“ in der Blockchain zu finden sind, sondern nur „Überweisungen“ – die Blockchain enthält also keinen

[4] In der Einführung zu verteilten Systemen in Abschnitt 2.6 wurden die an einem verteilten System Beteiligten, wie in der entsprechenden Literatur üblich, abstrakt als „Prozesse“ bezeichnet. Ab hier wird etwas konkreter von „Parteien“, „Beteiligten“ und „Teilnehmern“ gesprochen – damit wird klarer, dass in der Praxis Personen(gruppen) und deren Interessen das Verhalten der Prozesse bestimmen.

[5] Tatsächlich sind die ausgewählten Transaktionen bei Proof of Work bereits Teil des Rätsels, d. h. sie müssen vorher ausgewählt werden – das gilt aber nicht für alle Konsens-Algorithmen.

Zustand, sondern Zustandsübergänge[6]. Um den aktuellen Zustand explizit zu erhalten, müssen also – ausgehend von einem definierten Startzustand – alle Transaktionen in der richtigen Reihenfolge angewendet werden. Damit ist bereits klar, dass die Reihenfolge der Transaktionen eine wichtige Rolle spielt, wenn alle Teilnehmer den gleichen aktuellen Zustand „sehen“ sollen – wie das in einer Blockchain erreicht wird, ist in Abschnitt 3.2 beschrieben.

Abstrakt beschreibt eine Transaktion TX einen Zustandsübergang einer Datenstruktur. Dabei ist die Menge der möglichen Zustände, auf die TX angewandt werden kann, zunächst unbeschränkt. D. h. jede Transaktion kann auf jeden Zustand angewendet werden, das Ergebnis ist ein entsprechend veränderter Zustand. In einer Blockchain wird diese Zustandsmenge durch bestimmte – implementierungsabhängige – Regeln eingeschränkt, so ist z. B. in Bitcoin eine Transaktion nur dann „gültig“, wenn die Summe der ausgehenden Werte kleiner/gleich der Summe der eingehenden Werte ist[7].

Wegen der Grundannahme, dass sich die an der Blockchain beteiligten Parteien nicht vertrauen, werden Transaktionen in der Regel digital signiert. Dadurch ist z. B. für Transaktionen, bei denen Währungseinheiten transferiert werden, sichergestellt, dass nur der „Kontoinhaber“ Währungseinheiten von seinem „Konto“ versenden kann. In realen Implementierungen ist diese Signatur manchmal aufgeteilt: So wird z. B. bei Bitcoin die eigentliche Transaktion gar nicht signiert. Stattdessen muss eine Transaktion, die den Output einer anderen Transaktion „verbrauchen“ will, eine korrekte „Antwort“ auf eine Frage liefern; diese Frage ist Teil des „verbrauchten“ Outputs. In der Regel wird dabei nach einem bestimmten öffentlichen Schlüssel und einer Signatur, die mit dem zugehörigen privaten Schlüssel erstellt wurde, gefragt.

Da eine Blockchain nur Transaktionen speichert, muss es – um einen aktuellen Zustand berechnen zu können – einen definierten Startzustand geben. Dieser wird in einer Blockchain *Genesis Block* genannt und stellt in der Regel einen „normalen“ Block – nur eben ohne Vorgänger – dar, auf dessen Inhalt sich die zu Beginn beteiligten Parteien extern einigen[8]. Dieser Genesis Block muss allen Teilnehmern an der Blockchain aus vertrauenswürdiger Quelle bekannt sein. In der Praxis ist der Genesis Block meist direkt in der Software enthalten, die zur Teilnahme an der Blockchain benutzt wird – und dieser Software muss der Nutzer ohnehin vertrauen.

3.2 Blöcke

Wie in Abschnitt 3.1 erläutert, müssen die Transaktionen, die in einer Blockchain gespeichert werden, in eine definierte Reihenfolge gebracht werden. Die einfachste Möglichkeit wäre nun, für jede einzelne Transaktion eine Position in der Ausführungsreihenfolge der Transaktionen

[6] Alle aktuell relevanten Blockchain-Clients speichern den aus ihrer Sicht aktuellen Zustand trotzdem explizit, weil das große Performance-Vorteile bringt. Konzeptionell wäre das aber nicht notwendig, es würde reichen, die Transaktionen und deren Reihenfolge (also die Blöcke) zu kennen.

[7] Es gibt noch weitere Anforderungen und mit den Block Rewards auch eine Ausnahme von dieser Regel, diese Details werden später noch behandelt.

[8] „Einigen“ schließt hier stillschweigende Akzeptanz des von den Entwicklern vorgegebenen Blocks – wie z. B. bei Bitcoin – mit ein.

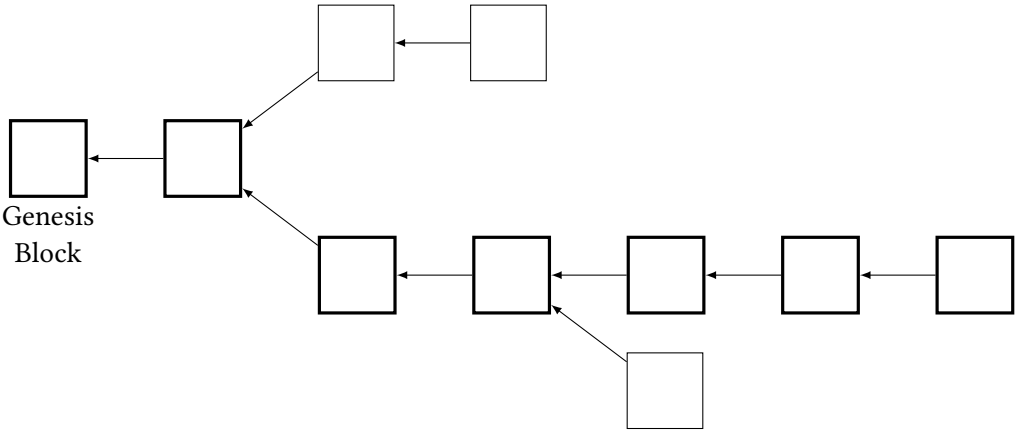


Abbildung 3.2: Baum von Blöcken einer Blockchain, die Pfeile stellen jeweils den Verweis auf den Vorgängerblock dar. Die fett hervorgehobenen Blöcke bilden den active Fork.

festzulegen. Da die Blockchain aber als verteiltes System sich gegenseitig nicht vertrauender Parteien entworfen wurde, müssten sich alle beteiligten Parteien für jede einzelne Position in der Ausführungsreihenfolge auf eine Transaktion „einigen“. In der Praxis führt das zu einem Problem: Eine solche Einigung dauert eine gewisse Zeit und bedeutet für die beteiligten Parteien einen gewissen Aufwand. Dieser Aufwand (und auch die nötige Zeit) ist allerdings fast unabhängig davon, auf wie viele Daten sich die Parteien einigen müssen. Daher ist es effizienter, Transaktionen zu Blöcken zu gruppieren und sich dann nur noch auf die Reihenfolge dieser Blöcke zu einigen.

Ein weiterer Vorteil der Verwendung von Blöcken ist, dass jeder Block seinen Vorgänger referenziert, dadurch ist die Reihenfolge der Blöcke ein impliziter Teil der Blockchain. Ein solcher Block besteht aus einem *Header* und einem *Body*. Der Blockheader enthält dabei mindestens einen Verweis auf den vorherigen Block sowie einen Hashwert über den Blockbody. Dazu kommen je nach Implementierung häufig ein Zeitstempel, eine Art „Kommentarfeld“, technisch notwendige Felder wie z. B. die Blockgröße sowie ggf. weitere Felder für den verwendeten Konsens-Algorithmus. Der Blockbody enthält dann die eigentlichen Transaktionen und deren Reihenfolge. Der angesprochene Hashwert über den Blockbody ist in der Regel als Hashbaum implementiert und schützt den Blockbody vor unbemerkten Veränderungen (siehe [80] zu Hashbäumen).

Verkettung der Blöcke Wie bereits erwähnt, enthält der Header eines Blocks einen Verweis auf seinen direkten Vorgänger, dieser Verweis ist in der Regel der Hashwert über den Blockheader des referenzierten Blocks. Der Grund dafür, dass dieser Hashwert nur über den Header (statt über den ganzen Block) gebildet wird, ist die einfachere Überprüfbarkeit: Um zu prüfen, ob ein Block tatsächlich zur längsten Kette gehört, genügt es, die relativ kleinen Blockheader zu kennen und deren Hashwerte zu berechnen. Sicherheit geht dabei nicht verloren, weil die

Header ja wiederum jeweils einen Hashwert über den Blockbody enthalten.

Durch diese Verweise auf den jeweiligen Vorgänger ergibt sich ein Baum von Blöcken, dessen Wurzel der Genesis Block ist. Abbildung 3.2 zeigt einen solchen Baum, gut erkennbar ist dabei, dass es *mehrere* Zweige geben kann, im Kontext von Blockchains nennt man diese Zweige *Forks*.

3.3 Mining

Da es – wie in Abbildung 3.2 dargestellt – vorkommen kann, dass es mehrere Forks aus gültigen Blöcken gibt, braucht es für eine einheitliche Sicht auf das Transaktionsverzeichnis auch ein gemeinsames Verständnis davon, welcher Fork der „richtige“ – der *active Fork* – ist. Bei Blockchains gilt dabei immer der *schwerste* Fork als active – unter der Annahme, dass alle Blöcke das gleiche Gewicht haben, wäre das also einfach der längste Fork. Daraus folgt, dass sich das Verständnis davon, was der active Fork ist, ändern kann, wenn ein vormals kürzerer Fork den aktuell längsten „überholt“.

Eines der Probleme, die dadurch entstehen können, ist in Abbildung 3.3 gezeigt: Alice will Mallory ein Auto für fünf Bitcoins verkaufen, Mallory willigt ein und sendet eine entsprechende Transaktion TX_2 an die Miner. Sobald Alice sieht, dass TX_2 in einen gültigen Block (hier B_2) aufgenommen wurde, übergibt sie das Auto an Mallory. Anschließend fährt Mallory davon und veröffentlicht die (gültigen) Blöcke B_3 und B_4, die er selbst erstellt hat und die eine Transaktion TX'_2 enthalten. TX'_2 „verbraucht“ dabei dieselbe Transaktion wie TX_2, d. h. nur eine von beiden kann ausgeführt werden. Weil der Fork B_1, B_3, B_4 der längste ist, wird er von den anderen Parteien als active Fork angenommen – Mallory hat sein Geld zurück und behält das Auto. Ein solcher Angriff wird *Double-Spending*-Angriff genannt, weil der Angreifer dasselbe Geld mehrfach ausgeben kann – im Beispiel gibt Mallory das Geld aus TX_1 zweimal aus.

Folglich muss verhindert werden, dass ein Angreifer ohne Einschränkungen gültige Blöcke erzeugen kann. Dafür werden in Blockchains sogenannte Konsens-Algorithmen verwendet. Ein solcher Konsens-Algorithmus muss mindestens die folgenden Eigenschaften erfüllen:

1. Das Erstellen *gültiger* Blöcke wird begrenzt bzw. mit (hohen) Kosten behaftet.
2. Falls diese Begrenzung auch die Blockrate steuern soll, muss sie einstellbar sein.
3. Es kann effizient geprüft werden, ob ein neuer Block gültig ist: Andernfalls können die Teilnehmer nicht ohne Weiteres entscheiden, welches die längste *gültige* Kette ist.

Seit der ersten Blockchain von Nakamoto wurden mehrere Konsens-Algorithmen entwickelt, die bekannteste (und auch bei Nakamotos Bitcoin verwendete) Technik ist der sogenannte *Proof of Work*, der im nächsten Abschnitt genauer betrachtet wird. Eine weitere inzwischen populäre Methode ist *Proof of Stake*[9], dabei wird anhand des „Vermögens“ auf der Blockchain

[9] Proof of Stake wird z. B. von Peercoin [67] verwendet, für Ethereum [37, 38] ist die Einführung einer Proof of Stake-Variante namens CASPER geplant, die bereits getestet wird.

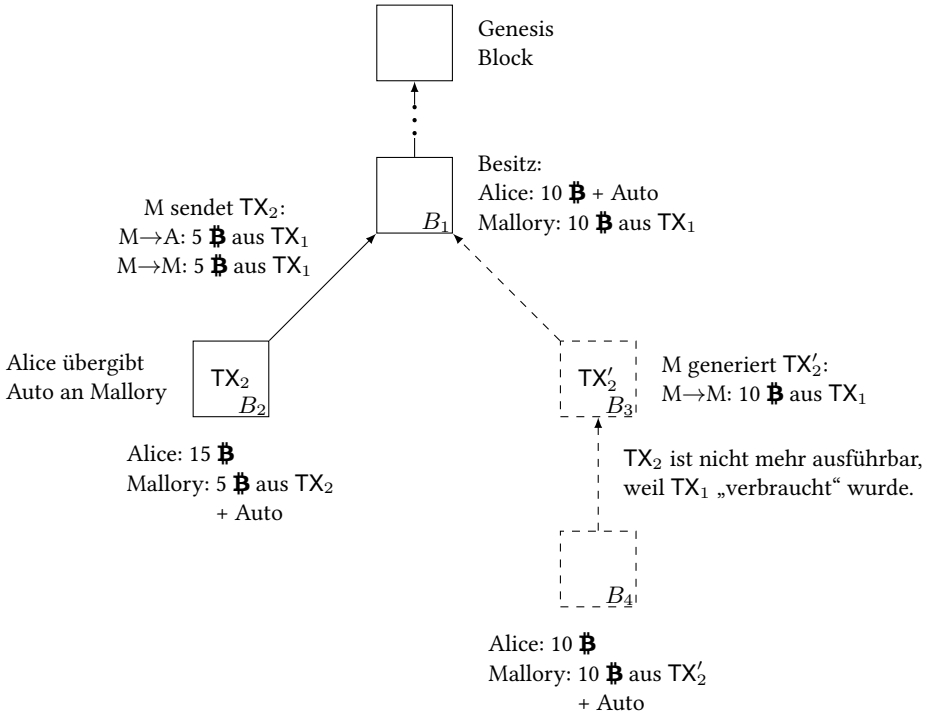


Abbildung 3.3: Double-Spending-Angriff: Alice sieht TX_2 in einem gültigen Block B_2 und übergibt Mallory ein Auto. Danach veröffentlicht Mallory die Blöcke B_3 und B_4, die mit TX'_2 eine Transaktion enthalten, durch die TX_2 nicht mehr ausführbar ist. Weil Mallorys Fork der längere ist, wird er zum active Fork und Mallory behält sein Geld *und* das Auto.

entschieden, wer den nächsten Block erstellen („*minen*“) darf – je höher das Vermögen, desto häufiger darf ein Block erstellt werden. Die Idee ist, dass Parteien mit großem Vermögen auf der Blockchain kein Interesse daran haben, durch Betrug den Ruf der Blockchain und damit den Wert ihres Vermögens zu gefährden. Ein verwandtes Verfahren ist der *Proof of Burn*[10], bei dem der Miner einen bestimmten Betrag pro erstelltem Block nachweislich „verbrennen“ muss – dieses Verfahren wird in der Regel verwendet, um eine neue Blockchain zu „starten“: Die Miner müssen Wert in einer bereits etablierten Währung „verbrennen“, um Blöcke für die neuen Blockchain erstellen zu dürfen. Weitere Verfahren sind *Proof of Elapsed Time*[11], *Proof of Space* bzw. *Proof of Capacity*[12] und *Practical Byzantine Fault Tolerance*[13]. Im Rahmen dieser Arbeit

[10] Proof of Burn kommt z. B. bei Slimcoin [88] (neben Proof of Work) zum Einsatz.

[11] Proof of Elapsed Time wurde ursprünglich von Intel entwickelt und wird im Hyperledger-Projekt Sawtooth [60] verwendet. Die Idee ist, dass die Auswahl des Miners für den nächsten Block innerhalb einer vertrauenswürdigen Laufzeitumgebung (z. B. Intel SGX) ausgeführt wird.

[12] Beim Proof of Space bzw. Proof of Capacity muss der Miner nachweislich eine gewisse Menge an Festplatten- oder Arbeitsspeicher belegen, um einen Block erstellen zu dürfen. Verwendung findet der Proof of Space z. B. in Burstcoin [45].

[13] Practical Byzantine Fault Tolerance ist ein Byzantine Agreement-Protokoll und wird z. B. im Hyperledger-Projekt Fabric [84] benutzt.

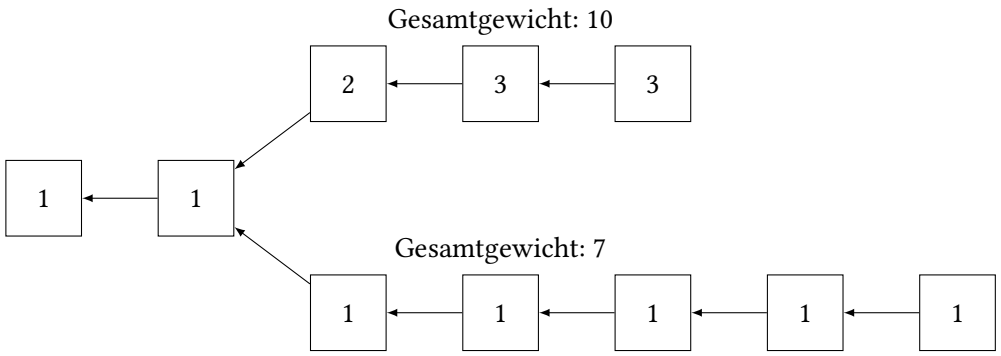


Abbildung 3.4: Beispiel zum Unterschied zwischen *längstem* und *schwerstem* Fork einer Proof of Work-Blockchain. In den Blöcken ist jeweils die Schwierigkeit des Blocks angegeben.

wird nur der Proof of Work näher beschrieben.

Unterscheidung „längste“ vs. „schwerste“ Kette Wie bereits erwähnt, ist das entscheidende Kriterium dafür, welcher Fork als active Fork gilt, nicht die Länge, sondern das „Gewicht“ des Forks. Ein Beispiel, welches diesen Unterschied verdeutlicht, ist in Abbildung 3.4 dargestellt: Der obere Fork ist zwar kürzer, die Miner mussten aber mehr Arbeit leisten, um ihn zu erstellen. Darum ist er eine sicherere Wahl, weil er schwerer zu „überholen“ ist – die dafür nötige Arbeit ist größer als beim unteren Fork. Zudem wird die Wahl des active Fork dadurch eine Art Abstimmung unter den Minern: Der Fork, an der die meisten Miner (bezogen auf ihre Rechenleistung) arbeiten, wird automatisch auch der schwerste bleiben oder werden.

Motivation der Miner Bisher unerwähnt geblieben ist die Frage, warum die Miner überhaupt am Mining teilnehmen. Da die Blockchain von einander nicht vertrauenden Parteien als Teilnehmer ausgeht, muss es eine Motivation für die Teilnahme am Mining geben, die nicht von Vertrauensverhältnissen außerhalb der Blockchain (etwa soziale Reputation) abhängt. Alle gängigen öffentlichen Blockchains implementieren deshalb eine eigene Währung. Wenn nun ein Miner einen Block erstellen darf, so erhält er in der Regel einen sogenannten *Block Reward*, also eine Belohnung dafür, dass er diesen Block erstellt hat. Dieser Block Reward entsteht dabei „aus dem Nichts“, meistens implementiert als eine spezielle Transaktion, die dem Miner den Block Reward gutschreibt. Neben dem Block Reward kann der Miner auch noch Transaktionsgebühren für alle Transaktionen, die er in den Block aufgenommen hat, einnehmen. Diese Transaktionsgebühren werden dabei nicht vom Miner festgelegt, sondern jede Transaktion weist einen vom Sender der Transaktion frei wählbaren Betrag als Transaktionsgebühr aus. Natürlich werden die Miner bei der Blockerstellung Transaktionen mit höheren Transaktionsgebühren bevorzugen.

3.4 Proof of Work

Um das Erstellen von gültigen Blöcken zu beschränken, wird bei Proof of Work-Blockchains von den Minern gefordert, dass sie ein rechenintensives Problem lösen, bevor sie einen Block erstellen dürfen. Wie dieses Problem genau aussieht, unterscheidet sich natürlich je nachdem, welche Blockchain untersucht wird. Das gängigste Verfahren ist dabei das 1997 von Adam Back für Hashcash [7] vorgeschlagene: Es soll ein Hashwert über eine definierte Menge an Daten (Blockheader) plus einige frei wählbare Bits gebildet werden. Die Schwierigkeit dabei ist nun, dass die ersten k Bits des resultierenden Hashwerts vorgegeben werden. Wird dabei eine kryptographische Hashfunktion verwendet, dann kann dieses Problem nur dadurch gelöst werden, dass so lange verschiedene Kombinationen der frei wählbaren Bits durchprobiert werden, bis die ersten k Bits mit der Vorgabe übereinstimmen. Der Parameter k macht es dabei möglich, die „Schwierigkeit“ der Aufgabe einzustellen.

Bei Blockchains wird das folgendermaßen genutzt: Im Blockheader wird ein zusätzliches Feld eingefügt[14], die Bits in diesem Feld darf der Miner frei wählen. Um die Vorgabe der ersten k Bits möglichst einfach zu halten, wird meistens nur gefordert, dass der Hashwert *kleiner* als ein Vorgabewert ist – d. h. die ersten k Bits müssen gleich Null sein. Wenn ein Miner also einen Block erstellen möchte, dann erstellt er zunächst den Blockbody und berechnet den Hashwert über die darin enthaltenen Transaktionen. Anschließend erstellt er den Blockheader mit dem Verweis auf den vorherigen Block, dem Hashwert über den Blockbody, ggf. zusätzlichen Feldern und dem frei belegbaren Feld, letzteres kann er z. B. zunächst mit Nullen füllen. Dann berechnet er den Hashwert über den so erstellten Blockheader – falls die ersten k Bits Null sind, kann er den Block veröffentlichen, andernfalls ändert er den Inhalt des frei belegbaren Felds, berechnet erneut den Hashwert über den Blockheader, prüft die ersten k Bits und so weiter. So ergibt sich eine Art Wettlauf zwischen den Minern, wer den nächsten Block erstellen darf – die Wahrscheinlichkeit, diesen Wettlauf zu gewinnen, ist dabei proportional zur Anzahl an Hash-Versuchen pro Sekunde, die ein Miner leisten kann.

Wäre der Parameter k fest, dann würde eine Zunahme der Hashleistung (über alle Miner addiert) dazu führen, dass Blöcke in immer kürzeren Abständen erzeugt werden. Eine immer weiter steigende Blockrate führt zu einer ganzen Reihe an Problemen, einige davon werden in Unterabschnitt 3.7.5 diskutiert. Um genau das zu verhindern, wird die Schwierigkeit an die Hashleistung aller Miner zusammen angepasst, sodass die Blockrate in etwa konstant bleibt. Der Algorithmus für diese Anpassung ist Teil der „Regeln“ einer Blockchain, d. h., damit ein Block als gültig anerkannt wird, muss die Schwierigkeit des Proof of Work nach diesem Algorithmus bestimmt worden sein.

[14] Dieses Feld wird meistens *Nonce* genannt. *Nonce* steht für „Number used once“, also eine Zahl, die nur einmal benutzt wird – in kryptographischen Protokollen werden Nonces häufig verwendet, um sicherzustellen, dass Nachrichten „frisch“ und nicht Aufzeichnungen aus vergangenen Protokollausführungen sind. Beim Proof of Work wird sie als „freies“ Feld verwendet, um möglichst einfach neue Blockheader-Varianten zu erzeugen.

3.5 Netzwerkstruktur

Da eine Blockchain ein verteiltes System bildet, ist klar, dass alle relevanten Daten – Blöcke, Transaktionen usw. – irgendwie an die Teilnehmer verteilt werden müssen. Auch an dieser Stelle hat die Grundannahme, dass sich die Teilnehmer gegenseitig nicht vertrauen, direkten Einfluss darauf, wie diese Verteilung erfolgen kann: Jede Form von zentralisiertem System – z. B. eine Client-Server-Architektur [33, S. 35] – ist damit ausgeschlossen, denn dafür müssten die Teilnehmer der zentralen Instanz vertrauen. Darum werden die Daten in Blockchains über ein *Peer-to-Peer-Netzwerk* ausgetauscht. Ein Peer-to-Peer-Netzwerk ist ein Netzwerk, in dem alle Beteiligten („Peers“) gleichberechtigt sind, d. h. insbesondere, dass es gibt keine zentralen Instanzen oder Kontrollorgane gibt [77, Kap. 1.3].

Seit den ersten Peer-to-Peer-Netzwerken Ende der 1990er/Anfang der 2000er wurden viele unterschiedliche Architekturen für Peer-to-Peer-Netzwerke entwickelt, meist mit dem Ziel, ein bestimmtes Datum im Netzwerk möglichst effizient zu finden und geschickt zu replizieren, sodass mit dem Ausstieg eines einzelnen Teilnehmers keine Daten verloren gehen, aber auch keine unnötige Redundanz vorhanden ist[15]. Die meisten öffentlichen Blockchains verwenden allerdings eine sehr simple Architektur, in der alle Teilnehmer einfach alle neuen Daten, die sie empfangen, an alle ihnen bekannten Teilnehmer weiterleiten. Der Vorteil dieser simplen Architektur ist, dass (absichtliches) Fehlverhalten einzelner Knoten, z. B. Zensurversuche, kaum Einfluss auf das Gesamtnetzwerk haben. Ein offensichtlicher Nachteil ist, dass jeder Peer alle Daten vorhalten muss – da sich die Parteien aber gegenseitig misstrauen, ist das ohnehin unumgänglich. Ein subtilerer Nachteil ist der Netzwerk-Overhead: Weil jeder Peer alle aus seiner Sicht neuen Nachrichten an alle ihm bekannten Peers weiterleitet, kommt es häufig vor, dass ein Peer dasselbe Datum – z. B. den gleichen Block – mehrfach erhält.

Um diesen Overhead etwas einzuschränken, gibt es in den meisten Standard-Softwares zu Blockchains einige „Filterregeln“, um zumindest offensichtlich ungültige Daten nicht weiterzuleiten – diese Filter sind aber nicht Teil der festen Regeln, die eingehalten werden müssen, die Anwendung ist also freiwillig.

Die Teilnehmer (im Sinne von Peers im Netzwerk) an einer Blockchain lassen sich grob in die folgenden drei Kategorien unterteilen, die sich nicht unbedingt gegenseitig ausschließen:

Full Nodes Eine *Full Node* verarbeitet (und speichert) alle erzeugten Blöcke seit dem Genesis Block und prüft dabei, ob die Gültigkeitsregeln beim Erzeugen der Blöcke eingehalten wurden und ob die darin enthaltenen Transaktionen gültig sind. Da nur die Full Nodes alle Gültigkeitsregeln prüfen, ist es für die Sicherheit einer Blockchain wichtig, dass möglichst viele der Teilnehmer eine solche Full Node betreiben. Andernfalls könnten die Miner – aus Versehen oder absichtlich – ungültige Blöcke erzeugen, z. B. mit höherem Block Reward als ihnen eigentlich zusteht. Aus Sicht des einzelnen Teilnehmers lohnt sich der Betrieb einer Full Node, weil nur so

[15] Ausführliche Diskussionen von Peer-to-Peer-Netzwerken und deren Entwicklung finden sich z. B. in [77] sowie [33, Kap. 10].

auf Vertrauen in andere Teilnehmer verzichtet werden kann – denn nur Full Nodes sind in der Lage, die Einhaltung aller Regeln zu prüfen.

Miner Wie bereits in Abschnitt 3.3 beschrieben, sind die Miner für das Erzeugen der Blöcke zuständig. Miner müssen nicht unbedingt zugleich auch Full Nodes sein, die Miner benötigen nur die Blockheader und die aktuell ausstehenden Transaktionen, um einen neuen Block zu erstellen[16]. In der Praxis ergibt es aber durchaus Sinn für einen Miner, auch eine Full Node zu betreiben – andernfalls könnte er nicht prüfen, ob die offenen Transaktionen, die er in einen Block einfügen möchte, überhaupt gültig sind – sollte ein Block eine ungültige Transaktion enthalten, so ist der ganze Block ungültig und wird vom Netzwerk abgelehnt, d. h. der Miner bekommt keinen Block Reward und auch keine Transaktionsgebühren.

Light Nodes Light Nodes verarbeiten nur die Teile der Blockchain, an denen sie interessiert sind, z. B. Transaktionen, die zu einem bestimmten „Konto“ gehören. Das bedeutet, dass sie zwar viel weniger Daten verarbeiten und herunterladen müssen, zugleich bedeutet es aber auch, dass Light Nodes nicht alle Gültigkeitsregeln prüfen können. Folglich müssen die Light Nodes den Peers, von denen sie ihre Informationen erhalten (z. B. Blockheader), zumindest zu einem gewissen Grad vertrauen. Zudem nehmen sie in der Regel nicht oder nur eingeschränkt am zugrunde liegenden Peer-to-Peer-Netzwerk teil, sondern beziehen ihre Informationen über direkte Verbindungen zu Full Nodes, die keinen direkten Anreiz haben, diese Informationen auch wirklich zu liefern. Dazu kommt noch, dass die Light Node dabei natürlich preisgeben muss, an welchen Informationen sie interessiert ist – Full Nodes dagegen haben ohnehin die komplette Blockchain lokal gespeichert, sie können daher alle für sie relevanten Informationen aus ihrer lokalen Kopie extrahieren.

3.6 Eine konkrete Implementierung: Bitcoin

Nachdem in den letzten Abschnitten die einzelnen Teile einer Blockchain recht abstrakt beschrieben wurden, wird in diesem Abschnitt eine konkrete Implementierung – Bitcoin – beschrieben. Bitcoin war die erste Blockchain überhaupt und wurde von Nakamoto hauptsächlich als dezentrale Zahlungsplattform entwickelt. Im Folgenden werden die Transaktionen, Blöcke, Mining und der Konsens-Algorithmus von Bitcoin beschrieben, das zugrunde liegende Peer-to-Peer-Netzwerk ist hier nicht weiter von Interesse. Es entspricht grob dem oben beschriebenen, bei dem jeder Peer alle für ihn neuen Informationen an alle ihm bekannten Peers weiterleitet.

[16] Das gilt zumindest für Proof of Work-Blockchains, es sind auch Konsens-Mechanismen denkbar, bei denen auch die Miner alle Blöcke benötigen – dennoch müssen die Miner keine Full Nodes sein, sie müssen die Blöcke schließlich nicht validieren.

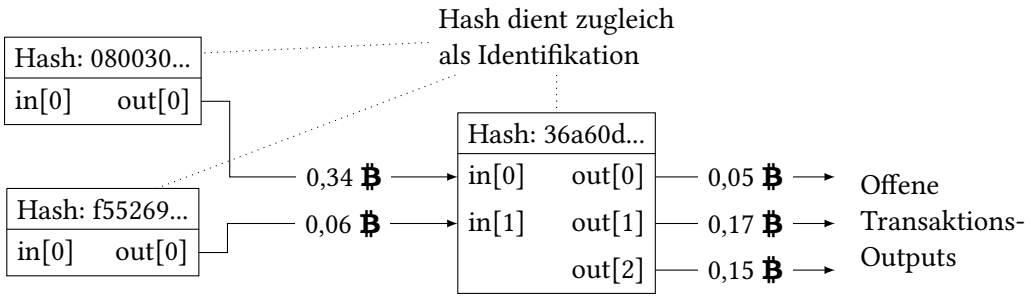


Abbildung 3.5: Schematische Darstellung einer Bitcoin-Transaktion mit zwei Inputs und drei Outputs. Die Differenz von 0,03 ฿ zwischen Input- und Outputsumme geht als Transaktionsgebühr an den Miner, der diese Transaktion in einen Block aufnimmt. Die Ausgänge out[0] der Transaktionen 0800... und f552... dürfen von keiner früheren Transaktion als Eingänge genutzt worden sein.

3.6.1 Transaktionen in Bitcoin

Da eine Blockchain nur Transaktionen speichert, muss für eine elektronische Währung wie Bitcoin ein Transaktionsmodell gewählt werden, bei dem möglichst einfach geprüft werden kann, ob eine Transaktion „gedeckt“ ist. Klassische Zahlungstransaktionen, die im Wesentlichen aus Quell-, Zielkonto und einem Betrag bestehen, sind dafür ungünstig: Um zu prüfen, ob eine Transaktion ausgeführt werden kann, müssten dafür nämlich alle Transaktionen seit Beginn der Blockchain durchlaufen werden, um den aktuellen Kontostand zu berechnen.

Nakamotos Lösung dafür ist, dass Transaktionen nicht von einem Konto auf ein anderes überweisen, sondern dass eine Transaktion als „Quellen“ die Ausgänge anderer Transaktionen angibt (siehe Abbildung 3.5 für ein Beispiel). Damit muss zur Prüfung einer Transaktion „nur“ noch geprüft werden, ob Ausgangswert $\leq$ Eingangswert und ob alle „verbrauchten“ Ausgänge noch zur Verfügung stehen; also noch nicht verbraucht wurden. Naiv erfordert auch diese Prüfung, dass alle vergangenen Transaktionen durchlaufen werden, allerdings gibt es zwei einfache Möglichkeiten, diesen Aufwand stark zu reduzieren: Zum einen kann der zu prüfende Ausgang erst *nach* Ausführung der zugehörigen Transaktion verbraucht werden, d. h. der Suchraum kann zeitlich eingegrenzt werden. Zum anderen können die Full Nodes eine Liste aller bisher unverbrauchten Transaktions-Ausgänge führen. Diese Liste enthält dann im Gegensatz zu einer Kontendatenbank – wie sie für klassische Zahlungstransaktionen möglich wäre – nur die tatsächlich noch nutzbaren Transaktions-Ausgänge und die Suche nach einem bestimmten Element in dieser Liste kann durch geschickte Implementierung (Hash-Listen) effizient ausgeführt werden.

Wie in Abschnitt 3.1 bereits erwähnt, hat eine Transaktion in Bitcoin nicht nur jeweils einen Ein- und Ausgang, sondern kann mehrere Ein- und Ausgänge haben, die üblicherweise mit ihren englischen Namen *Inputs* und *Outputs* bezeichnet werden. Abbildung 3.5 zeigt schematisch eine solche Transaktion. Die Outputs enthalten zum einen den Betrag und zum anderen den Hashwert eines öffentlichen Schlüssels – wer diesen Output später verbrauchen will, muss einen

Feldname	Beispielwert	Feldgröße	Beschreibung
Version	4	4 Bytes	Protokollversion
Previous	`0x5cfe6cd...`	32 Bytes	Hash des Vorgänger-Blockheaders
Merkle Root	`0xabfa0a2...`	32 Bytes	Wurzel des Hashbaums über den Blockbody
Timestamp	`0x5a69dd7c`	4 Bytes	Zeitstempel der Blockerstellung
Bits	`0x1337beef`	4 Bytes	Schwierigkeit des Proof of Work (spezielles Format)
Nonce	827263	4 Bytes	Vom Miner frei wählbar, um Proof of Work zu lösen

Tabelle 3.1: Felder im Bitcoin-Blockheader

solchen Schlüssel liefern, der zum Hashwert passt und mit dem zugehörigen geheimen Schlüssel eine Signatur erstellen. Die Inputs enthalten dabei die Information, welcher Output durch diesen Input verbraucht wird, d. h. für den Input in[1] wäre das der nullte Output der Transaktion „f55269...".Dazu kommt noch eine Signatur, die bestätigt, dass der Sender der Transaktion auch berechtigt ist, den spezifizierten Output zu verbrauchen[17]. Wie in der Abbildung zu sehen, werden die Transaktionen an sich – analog zu Blöcken – über einen Hashwert identifiziert.

3.6.2 Blöcke und Mining in Bitcoin

Ein Bitcoin-Block besteht aus einem Blockheader und einem Blockbody. Tabelle 3.1 zeigt die Felder des Blockheaders mit jeweils einer kurzen Beschreibung. Neben den beiden in jeder Blockchain nötigen Hash-Verweisen auf Vorgänger-Block und Blockbody enthält der Blockheader bei Bitcoin folgende Felder: Eine Versionsnummer, um eine gewisse Weiterentwicklung zu ermöglichen, einen vom Miner vergebenen Zeitstempel, die aktuelle Schwierigkeit des Proof of Work sowie eine Nonce, die ebenfalls für den Proof of Work-Konsens-Algorithmus benötigt wird.

Der Blockbody enthält eine (geordnete) Liste von Transaktionen, die erste dieser Transaktionen ist die sogenannte *Coinbase*-Transaktion. In der Coinbase-Transaktion zahlt der Miner, der den Block erstellt hat, sich selbst den Block Reward plus die Transaktionsgebühren der im Block enthaltenen Transaktionen aus. Der Block Reward für Bitcoin-Blöcke war zu Beginn 50 Bitcoins pro Block und wird alle 210.000 Blöcke[18] halbiert. Die inneren Knoten des Hashbaums über die Transaktionen sind *nicht* Teil eines Blocks, sondern müssen zur Validierung des Blocks neu berechnet das Ergebnis (die Wurzel) und mit dem Wurzelhash im Blockheader vergleichen werden – das wäre aber ohnehin notwendig, sonst könnte der Miner unbemerkt einen ungültigen

[17] Hier verwendet Bitcoin sogenannte Scripts: Kleine Programme in einer einfachen Programmiersprache. In- und Output liefern jeweils einen Teil des Programms und damit die Transaktion gültig ist, muss das Programm ohne Fehler terminieren und einen Wert ungleich Null zurückgeben. Für einfache Bitcoin-Transfers prüft dieses Script einfach eine Signatur. Im Rahmen dieser Einführung werden Scripts nicht näher behandelt, Details finden sich z. B. in [82].

[18] 210.000 Blöcke entsprechen bei einer Blockrate von ca. einem Block alle zehn Minuten etwa vier Jahren.

Hashbaum in einen Block einbauen. Dank der festen Reihenfolge der Transaktionen in einem Block und einem festgelegten Algorithmus zum Aufbau des Baums braucht es auch keine weitere Information darüber, wie der Baum genau gebildet wird.

Proof of Work und Mining Bei Bitcoin wird als Konsens-Algorithmus ein Proof of Work verwendet, bei dem der Blockheader doppelt mit einer Hashfunktion namens SHA-256 [93] gehasht wird – der resultierende Hashwert muss dann kleiner als ein vorgegebener Wert sein. Da SHA-256 eine kryptographische Hashfunktion ist, müssen die Miner, wie in Abschnitt 3.4 beschrieben, so lange leicht veränderte Varianten des Blockheaders erzeugen und hashen, bis das Ergebnis „klein genug“ ist. Dafür enthält der Blockheader das *Nonce*-Feld: Üblicherweise erstellt der Miner einen Blockbody, berechnet den Hashbaum und erstellt dann einen zugehörigen Blockheader, bei dem die Nonce gleich Null ist. Dann wird der Header gehasht – ist das Ergebnis klein genug, kann der Block veröffentlicht werden. Andernfalls – und meistens – ist der Hashwert aber zu groß, dann ist die schnellste Möglichkeit, einen neuen Wert zu testen, die Nonce zu verändern und erneut den Hashwert zu berechnen. In der Regel wird die Nonce dabei einfach inkrementiert, der Miner kann aber auch andere Strategien wählen.

Nun gibt es für die Nonce nur 2^{32} Möglichkeiten[19]. Wenn diese ausgeschöpft sind, muss der Miner einen anderen Teil des Blockheaders ändern, das kann z. B. der Zeitstempel sein oder er verändert den Blockbody und bekommt so einen neuen Hashwert als Wurzel des Hashbaums – dann kann er wieder mit Nonce gleich Null beginnen und so weiter.

Um die Blockrate unabhängig von der insgesamt eingesetzten Hashleistung ungefähr konstant zu halten, wird die Schwierigkeit des Proof of Work alle 2.016 Blöcke[20] angepasst. Dazu wird die durchschnittliche Blockrate der letzten 2.016 Blöcke berechnet, war sie höher als sechs Blöcke pro Stunde, dann wird die Schwierigkeit erhöht; war sie kleiner als sechs Blöcke pro Stunde, dann wird die Schwierigkeit reduziert. Da die letzten 2.016 Blöcke allen Teilnehmern bekannt sind (andernfalls würden sie den aktuellen Block gar nicht akzeptieren), kann jeder Teilnehmer selbst berechnen, was der neue Wert für die Schwierigkeit sein muss und so wiederum die Berechnung der Miner verifizieren.

3.7 Limitierungen und Probleme von Bitcoin und Blockchains allgemein

Neben den genannten positiven Eigenschaften hat Bitcoin auch einige negative Eigenschaften bzw. Limitierungen, die wegen der weiten Verteilung und dem gegenseitigen Misstrauen der Teilnehmer auch nicht ohne Weiteres behoben werden können bzw. zum Teil gerade deswegen notwendig sind. Außerdem müssen inkompatible Änderungen von einer großen Mehrheit der Teilnehmer akzeptiert werden, andernfalls würde sich die Blockchain in zwei inkompatible

[19] „Nur“ 2^{32} Möglichkeiten deshalb, weil bei der aktuellen Schwierigkeit (April 2018) des Proof of Work im Mittel mehr als 2^{70} Versuche nötig sind, um einen gültigen Block zu finden.
[20] 2.016 Blöcke entsprechen etwa zwei Wochen.

Forks teilen, die auch in Zukunft nicht mehr zusammengeführt werden können, weil der eine Fork die Blöcke des anderen nicht als gültig anerkennt. Eine solche inkompatible Änderung der Regeln wird als *Hard Fork* bezeichnet – in Abgrenzung zum *Soft Fork*, bei dem Blöcke und Transaktionen, die nach den neuen Regeln erstellt wurden, bezüglich der alten Regeln ebenfalls gültig sind; umgekehrt muss das nicht unbedingt der Fall sein. Im Folgenden werden einige der Nachteile bzw. Limitierungen von Bitcoin beschrieben, dabei werden soziale und einige der ökonomischen Aspekte bewusst ausgeklammert, diese werden z. B. in [82] ausführlicher diskutiert.

3.7.1 Transaktionsrate

Für die Blöcke in Bitcoin gibt es eine maximale Größe, die nur durch einen Hard Fork geändert werden kann, aktuell liegt sie bei einem Megabyte. Das führt dazu, dass nur eine begrenzte Anzahl Transaktionen in einen Block aufgenommen werden kann. Abhängig von der Größe der Transaktionen (z. B. Anzahl In-/Outputs) passen in einen Block ca. 1.000–5.000 Transaktionen. Da die Blockrate durch die Anpassung der Schwierigkeit des Proof of Work relativ konstant bei ca. sechs Blöcken pro Stunde liegt, ergibt sich eine Transaktionsrate zwischen zwei und acht Transaktionen pro Sekunde. Große Zahlungsdienstleister wie Visa und PayPal verarbeiten pro Sekunde tausende Transaktionen, Bitcoin ist also mehrere Größenordnungen langsamer. Daraus folgt, dass Bitcoin als alltägliches, weit verbreitetes Zahlungsmittel in seiner aktuellen Form nicht funktionieren kann.

Andere Blockchains erlauben zwar durch größere Blöcke oder höhere Blockraten deutlich mehr Transaktionen pro Sekunde, diese Erhöhung hat aber unerwünschte Nebenwirkungen, welche in Unterabschnitt 3.7.5 noch diskutiert werden. Ein bekanntes Beispiel ist Ethereum, dort wird im Mittel alle 15 Sekunden ein neuer Block erzeugt – diese Blöcke sind zwar kleiner, die resultierende Transaktionsrate ist dennoch deutlich höher; von tausenden Transaktionen pro Sekunde aber noch immer weit entfernt.

3.7.2 Timestamps bei Proof of Work-Blockchains

Der Timestamp in den Blockheadern hat einen wichtigen Einfluss auf die Schwierigkeit des Proof of Work und damit auf die Sicherheit der Blockchain, zugleich kann der Timestamp aber nicht so „streng“ validiert werden wie der Rest der Daten. Das liegt zum einen daran, dass der Block vom Miner bis zu dem Teilnehmer, der den Block validieren möchte, eine gewisse Zeit im Netzwerk „unterwegs“ ist; zum anderen laufen Uhren von unterschiedlichen Systemen nie ganz synchron. Vor allem aber sorgt schon das Minen an sich für eine gewisse Streuung der Block-Abstände: die genaue Dauer, bis ein gültiger Block gefunden wird, hängt vom „Glück“ der Miner ab. Zwar werden verschiedene Heuristiken verwendet, um einer Manipulation vorzubeugen, trotzdem muss den Timestamps zu einem gewissen Grad vertraut werden. Das ist im Sinne des gegenseitigen Misstrauens ein Problem, weil die Miner eine Motivation hätten, die Schwierigkeit zu beeinflussen: Wenn die Schwierigkeit sinkt, können die Miner mit gleichem Einsatz

mehr Blöcke minen und dadurch mehr Block Rewards (und evtl. auch Transaktionsgebühren) einnehmen.

3.7.3 ASIC-Mining

Die in Bitcoin verwendete Hashfunktion SHA-256 lässt sich gut auf spezialisierte Hardware, sogenannte *Application-Specific Integraced Circuits* (ASICs), abbilden. Da diese ASICs bezüglich der Mining-Leistung mehrere Größenordnungen schneller als normale Prozessoren sind, lohnt sich das Mining auf „normalen" Computern nicht mehr, seit es entsprechende ASICs gibt. Das führte und führt dazu, dass es immer weniger einzelne Miner gibt, weil sich der Kauf und Betrieb von ASICs vor allem in größeren Maßstäben lohnt – im Gegensatz zu normalen Computern, die ohnehin weit verbreitet vorhanden sind. In der Folge wird das gesamte System zentralisierter, was der ursprünglichen Idee eines komplett dezentralen Systems widerspricht.

Als Reaktion auf diese Entwicklung bei Bitcoin wurden für andere Proof of Work-Blockchains andere Hashfunktionen gewählt, bei denen der potenzielle Geschwindigkeitsgewinn durch ASICs deutlich kleiner ausfällt. Ein Beispiel ist auch hier wieder Ethereum, der dort verwendete Konsens-Algorithmus *Ethash*[21] wurde mit dem Ziel entworfen, möglichst viel Speicherbandbreite auszunutzen – diese ist in gewöhnlichen Grafikkarten schon sehr hoch, was den Einsatz von ASICs wirtschaftlich uninteressanter macht[22].

Dieses Problem betrifft natürlich in erster Linie Proof of Work-Blockchains, aber auch für einige der anderen Konsens-Algorithmen wie z. B. Proof of Space sind spezialisierte Hardwarelösungen denkbar, wenngleich der Vorteil gegenüber „normalen" Computern kleiner ausfallen dürfte. Ganz vermeiden lässt sich das Problem z. B. durch die Verwendung von Proof of Stake – die benötigte Rechenleistung dafür kann jedes moderne Smartphone zur Verfügung stellen.

3.7.4 Energieverbrauch durch Mining bei Proof of Work-Blockchains

Die große Anzahl an Hashwerten, die beim Mining durch die Miner immer wieder berechnet werden muss, führt zu einem hohen Energieverbrauch, der inzwischen in der Größenordnung eines ganzen Landes liegt[23]. Aus dieser Beobachtung heraus sind Vorschläge für alternative Proof of Work-Algorithmen entstanden, bei denen die Lösung des Proof of Work-Problems neben dem Erstellen eines Blocks noch anderweitig genutzt werden kann. Ein Beispiel für eine Implementierung eines solchen *Proof of useful Work* stellt Primecoin dar: Die Miner müssen

[21] Zur Klarstellung: Ethash ist der Konsens-Algorithmus und nicht – wie der Name suggeriert – die verwendete Hashfunktion. Als Hashfunktion wird in Ethereum *KECCAK-256* [13] verwendet.

[22] Aber nicht unmöglich: Ein ASIC für Ethereum wurde Anfang April 2018 angekündigt, ob der Betrieb auch rentabel wäre, ist bislang unklar [63].

[23] Schon 2014 wurde der Energieverbrauch durch Bitcoin-Mining von O'Dwyer und Malone [85] auf etwa 3 GW geschätzt – was in etwa dem Stromverbrauch von Irland entspricht. In einer neueren Veröffentlichung aus 2017 schätzt Hayes [54] die Leistungsaufnahme pro Gigahash/Sekunde (GH/s) für die effizientesten ASICs auf 0,15W pro GH/s. Bei der aktuellen (April 2018) Gesamthashrate von knapp 30 Milliarden GH/s [15] ergibt das eine Leistungsaufnahme von rund 4,4 GW – was in etwa dem Strombedarf Polens [75] entspricht.

möglichst lange Ketten von bestimmten Primzahlpaaren finden, diese werden dann z. B. in der Mathematik-Forschung verwendet. Neben Primecoin gibt es noch eine ganze Reihe an Vorschlägen, bei denen die eingesetzte Rechenleistung „echte“ Probleme quasi nebenbei lösen soll – eine große Hürde dabei ist aber, dass die Schwierigkeit des Problems sinnvoll einstellbar sein muss. Auch hier ist ein möglicher Ausweg die Verwendung anderer Konsens-Mechanismen, so soll z. B. Ethereum schrittweise von Proof of Work auf Proof of Stake umgestellt werden.

3.7.5 Blockchain Trilemma

Ähnlich dem CAP-Theorem (siehe Unterabschnitt 2.6.2) gibt es auch bei Blockchains drei grundlegende Eigenschaften, von denen eine Blockchain nur zwei zugleich erfüllen kann. In der (recht jungen) Literatur dazu [19, 78, 100] gibt es verschiedene Bezeichnungen für die Eigenschaften, gemeint ist aber im Kern dasselbe (siehe [52]). In Anlehnung an andere aktuelle Arbeiten [52, 108], die das Thema behandeln, werden hier die Bezeichnungen und Definitionen von Buterin aus der „Ethereum Sharding FAQ“ [19] verwendet:

- **Dezentral** Das System kann so betrieben werden, dass jedes einzelne „Endgerät“ nur über begrenzte Ressourcen verfügt. So wird sichergestellt, dass das Gesamtsystem aus vielen „normalen“ Computern bestehen kann und nicht nur aus einigen wenigen Teilnehmern mit besonders leistungsfähigen Geräten.
- **Skalierbar** Das Gesamtsystem kann bezüglich der Transaktionsrate skaliert werden – im Idealfall kann das Gesamtsystem mehr Transaktionen verarbeiten als jeder einzelne Teilnehmer.
- **Sicher** Das System ist sicher gegen einen Angreifer, dessen Ressourcen kleiner als das Gesamtsystem sind.

Die Spannung zwischen diesen Eigenschaften besteht nun darin, dass z. B. Dezentralität eine begrenzte Blockrate und -größe erfordert, zugleich müssten diese für eine skalierbare Transaktionsrate aber wachsen können. Blöcke und Transaktionen irgendwie parallel – je durch einen Teil des Gesamtsystems – zu verarbeiten, würde wiederum die Sicherheit reduzieren.

Weiter haben Blockrate und -größe auch einen Einfluss auf die Sicherheit, der z. B. von Gervais et al. [47] untersucht wurde: Weil mit höherer Blockrate/-größe die Übertragungszeit im Peer-to-Peer-Netzwerk relativ zur Zeit zwischen zwei Blöcken größer wird, gibt es mehr *stale Blocks*, also Blöcke, die zwar gültig, aber nicht im active Fork enthalten sind. Diese stale Blocks sind für die Sicherheit nachteilig, weil dabei ein Teil der Gesamtleistung des Systems in Blöcke gesteckt wird, die dann „weggeworfen“ werden. Dementsprechend bräuchte ein Angreifer weniger Ressourcen, um den gleichen Anteil an gültigen Blöcken *im active Fork* zu „erhalten“.

3.7.6 Eclipse Attacks

Ein mit Blick auf die Eigenschaften einer Blockchain besonders interessantes Angriffsszenario stellen die sogenannten *Eclipse Attacks* dar [47, 56, 83, 99, 107]. Bei einer Eclipse Attack

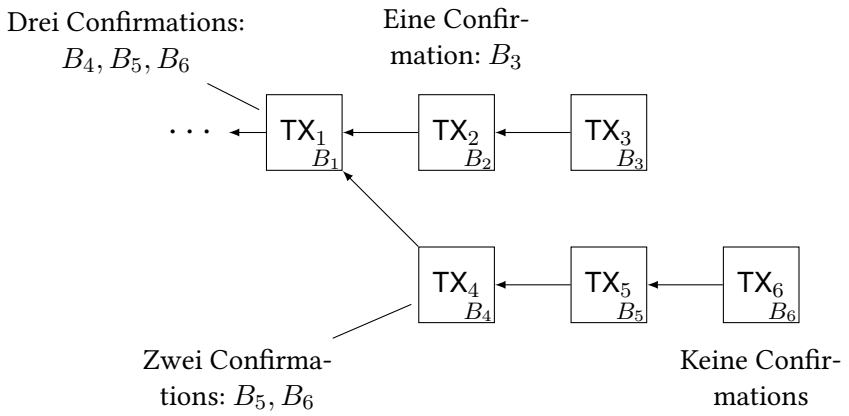


Abbildung 3.6: Beispiele für Confirmations von Transaktionen und Blöcken

„schneidet“ der Angreifer einen Teilnehmer einer Blockchain vom Rest des Netzwerks ab, d. h. er kontrolliert alle ein- und ausgehenden Verbindungen des Opfers. Dadurch kann der Angreifer dem Opfer andere Informationen (Blöcke, Transaktionen) präsentieren, als sie der Rest des Netzwerks sieht.

Eine Eclipse Attack eröffnet dem Angreifer eine ganze Reihe von Möglichkeiten: Er kann relativ einfach Double-Spending-Angriffe ausführen (selbst dann, wenn das Opfer noch einige Blöcke nach jenem, der die eigentliche Transaktion enthält, wartet); er kann die Rechenleistung des Opfers „missbrauchen“, um Angriffe auf andere Opfer auszuführen (vgl. [56, 83]); er kann dem Opfer „Zahlungseingänge“ vorenthalten und noch einiges mehr.

Das konzeptionelle Problem mit Eclipse Attacks ist, dass die Regeln einer Blockchain einen solchen Angriff nicht verhindern können. D. h., dass viele der Eigenschaften einer Blockchain nur unter der Annahme erfüllt werden, dass der Angreifer nicht in der Lage ist, eine Eclipse Attack auszuführen. Wie Heilman et al. in [56] beschreiben, ist diese Annahme aber zumindest für Bitcoin kaum haltbar. Durch verschiedene Techniken kann eine Eclipse Attack aber zumindest mit hoher Wahrscheinlichkeit erkannt werden, z. B. indem Blockrate und -schwierigkeit überwacht werden und bei größeren Abweichungen manuell eingegriffen wird. Auf Kosten des gegenseitigen Misstrauens aller Teilnehmer kann auch mit Listen von vertrauenswürdigen Peers gearbeitet werden (siehe [56]).

3.8 Eigenschaften von Blockchains

Bevor hier die später benötigten Eigenschaften von Blockchains vorgestellt werden, wird der Begriff der *Confirmations* eingeführt: Wie in Abbildung 3.6 dargestellt, hat ein Block oder eine Transaktion in einem Block c Confirmations, wenn es aufbauend auf diesem Block eine Kette aus c gültigen Blöcken gibt.

Persistenz Die erste Eigenschaft, die hier betrachtet wird, ist die *Persistenz*: Eine Blockchain erfüllt Persistenz, wenn eine Transaktion, die bei einem ehrlichen Teilnehmer mindestens c Confirmations hat, mit sehr hoher Wahrscheinlichkeit in den active Fork jedes ehrlichen Teilnehmers aufgenommen wird und eine feste Position in der Abfolge der Transaktionen bekommt [44].

Für diese Arbeit ist dabei besonders interessant, dass die dafür notwendige Anzahl an Confirmations nicht nur davon abhängt, welcher Anteil der gültigen Blöcke durch den Angreifer erzeugt wird, sondern auch davon, wie groß diese Blöcke sind, wie hoch die Blockrate ist und wie lange es dauert, bis alle Peers des zugrunde liegenden Peer-to-Peer-Netzwerks einen neuen Block erhalten haben [47]. Denn daraus folgt die relativ niedrige Transaktionsrate aktuell populärer Blockchains.

Lebendigkeit Eine Blockchain erfüllt Lebendigkeit, wenn jede Transaktion eines ehrlichen Teilnehmers in endlicher Zeit mindestens c Confirmations im active Fork eines ehrlichen Teilnehmers bekommt. Zusammen mit der Persistenz-Eigenschaft folgt daraus, dass jede Transaktion eines ehrlichen Teilnehmers mit sehr hoher Wahrscheinlichkeit in den active Fork jedes ehrlichen Teilnehmers aufgenommen wird.

(Fast) synchrones Netzwerk Um die Eigenschaften Persistenz und Lebendigkeit erfüllen zu können, muss das Verhältnis von Nachrichtenverzögerung im Netzwerk und Dauer zwischen zwei Blöcken ausreichend klein sein [44]. Das hängt mit der für Persistenz geforderten *sehr hohen* Wahrscheinlichkeit zusammen und kann etwas einfacher mit Blick auf das CAP-Theorem (siehe Unterabschnitt 2.6.2) verstanden werden. Bei einer Blockchain bleibt nämlich – entgegen dem intuitiven ersten Eindruck – während einer Netzwerkpartitionierung nicht Consistency, sondern Availability erhalten: Jede Netzwerkpartition arbeitet an „ihrem“ Fork unabhängig von den anderen Partitionen weiter und könnte so bei ausreichend lang anhaltender Partitionierung auch c oder mehr gültige Blöcke erzeugen.

Ehrliche Mehrheit der Miner Nicht direkt eine Eigenschaft der Blockchain selbst, aber eine Einschränkung des Angreifermodells: Die Mehrheit der gültigen Blöcke muss durch ehrliche Miner erzeugt werden – andernfalls können die Eigenschaften Persistenz und Lebendigkeit nicht eingehalten werden. Das kann man sich intuitiv klar machen: Wenn der Angreifer in einem festen Zeitraum mehr Blöcke erzeugen kann als alle ehrlichen Miner zusammen, dann kann er so lange Blöcke erzeugen und geheim halten, bis die ehrlichen Miner für einen bestimmten Block c Confirmations erzeugt haben – in dieser Zeit hat der Angreifer aber sehr wahrscheinlich einen schwereren Fork erzeugt. Diesen kann er nun veröffentlichen – weil dieser schwerere Fork zum neuen active Fork wird, werden Blöcke mit c Confirmations „ersetzt“ und so die Persistenz-Eigenschaft verletzt. Nimmt der Angreifer in die von ihm erzeugten Blöcke zudem nicht alle Transaktionen auf, wird auch Lebendigkeit verletzt.

4 Bulletin Boards

Ein Bulletin Board ist im Kern genau das, was der Name Bulletin Board – zu deutsch „schwarzes Brett“ – vermuten lässt: Eine öffentliche „Pinnwand“, an der Informationen veröffentlicht und anschließend von allen Interessenten gelesen werden können. Analoge Bulletin Boards bieten dabei jedem Leser – unter der Annahme, dass keine Informationen entfernt werden – einen *Consistent View* auf die vorhandenen Informationen. Ein Consistent View bedeutet, dass jeder Leser die gleichen Informationen sieht. Die einzige Ausnahme von dieser Regel sind Informationen, die zwischen den Lesevorgängen neu hinzugekommen sind. Bei digitalen Bulletin Boards ist es dagegen nicht selbstverständlich, dass jeder Leser dasselbe sieht. Das gilt auch dann, wenn die Lesevorgänge genau gleichzeitig stattfinden, denn das digitale Bulletin Board könnte einfach jedem Leser eine andere „Version“ der Informationen ausliefern.

Der Rest dieses Kapitels behandelt daher digitale Bulletin Boards und deren Eigenschaften und ist wie folgt aufgebaut: Zunächst werden anhand von Veröffentlichungen, in denen Bulletin Boards verwendet oder entworfen werden, die wichtigsten Eigenschaften eines möglichst allgemeinen Bulletin Boards identifiziert. Anschließend werden – motiviert durch konkrete Angriffsszenarien – verschiedene Maßnahmen vorgestellt, mittels derer die zuvor identifizierten Eigenschaften erfüllt werden können. Eine ausführliche Diskussion bestehender Vorschläge und Implementierungen von Bulletin Boards schließt das Kapitel ab.

4.1 Anforderungen an Bulletin Boards

Durch die Vielzahl an vorgeschlagenen Anwendungen für Bulletin Boards – insbesondere eVoting-Systeme – ergeben sich eine ganze Reihe von unterschiedlichen Anforderungen an ein Bulletin Board. In diesem Abschnitt werden einige Arbeiten vorgestellt, in denen Bulletin Boards verwendet, definiert oder direkt entworfen werden. Der Fokus liegt dabei auf den Forderungen, die darin an ein Bulletin Board gestellt werden[1]. Diese Forderungen werden zugunsten der Verständlichkeit und Übersichtlichkeit in idealisierter Form vorgestellt, d. h. ohne Berücksichtigung der Voraussetzungen, unter denen sie erfüllt werden können oder sollen. Ein Beispiel für diese Idealisierung ist, dass hier nicht darauf eingegangen wird, welcher Anteil der Peers eines verteilten Bulletin Boards kompromittiert sein darf – oder im Fall eines zentralen Bulletin Boards: Ob das Bulletin Board kompromittiert sein darf, um die jeweilige Eigenschaft noch zu erfüllen. Aus diesem Überblick wird dann eine Reihe von Anforderungen abgeleitet, die ein Bulletin Board erfüllen sollte.

[1] Das bedeutet insbesondere, dass hier *nicht* darauf eingegangen wird, welche der Eigenschaften ein bestimmtes Bulletin Board erfüllt oder nicht – diese Betrachtung folgt in Abschnitt 4.3.

Die Bezeichnungen der jeweils geforderten Eigenschaften weichen zum Teil von denen in den beschriebenen Arbeiten ab, um zu verdeutlichen, welche Eigenschaften ähnlich oder sogar äquivalent sind.

4.1.1 Häufige Anforderungen

In diesem Abschnitt werden die häufiger geforderten Eigenschaften kurz eingeführt, in den folgenden Abschnitten wird dann nur noch angegeben, ob und ggf. mit welchen Abweichungen diese gefordert werden. Dabei ist zu beachten, dass für viele der Eigenschaften hier „nur“ gefordert wird, dass dem Bulletin Board ein mögliches Fehlverhalten *nachgewiesen* werden kann (anstatt einfach die Einhaltung der Eigenschaft zu fordern). Der Grund dafür ist, dass für die garantierte Einhaltung in der Regel sehr starke Annahmen notwendig sind, die in der Praxis ohnehin kaum oder gar nicht erfüllt werden können[2]. Davon abgesehen sind die Eigenschaften weitgehend idealisiert formuliert, d. h. ohne Voraussetzungen, Angreifermodell o. ä. zu definieren. Eine genauere Betrachtung der notwendigen Voraussetzungen für verschiedene Bulletin Boards findet später in Abschnitt 4.3 statt.

Append-Only Einmal veröffentlichte Nachrichten werden nicht mehr entfernt oder verändert – das schließt auch „Metadaten“ wie Zeitstempel, Absenderidentität oder Sequenznummern ein. Sollte sich das Bulletin Board nicht daran halten, dann muss es dafür einen Beweis geben, der durch Dritte verifizierbar ist. Das muss aus Sicht eines einzelnen Lesers/Beobachters des Bulletin Boards gelten, damit Append-Only als erfüllt gilt.

Receipts Wenn ein Nutzer eine Nachricht an das Bulletin Board sendet, die vom Bulletin Board akzeptiert wird, dann muss dieser Nutzer eine „Empfangsbestätigung“ erhalten, im Folgenden Receipt genannt. Mit einem solchen Receipt kann der Nutzer dem Bulletin Board Fehlverhalten nachweisen, falls die zugehörige Nachricht nicht oder in veränderter Form veröffentlicht wird.

Consistent View Alle Leser des Bulletin Boards „sehen“ dasselbe. D. h. insbesondere, dass sobald ein Leser eine Nachricht m auf dem Bulletin Board „sieht“, auch alle anderen Leser ab diesem Zeitpunkt m auf dem Bulletin Board „sehen“. Auch hier gilt wieder: Falls das Bulletin Board verschiedenen Lesern unterschiedliche Daten anzeigt, muss es dafür einen Nachweis geben (ggf. müssen die Leser zusammenarbeiten, um diesen Nachweis zu erbringen).

[2] Z. B. müssten *alle* Peers eines verteilten Bulletin Boards ehrlich sein, damit beim Lesen von einem dieser Peers garantiert Append-Only eingehalten wird. Soll dagegen „nur“ Fehlverhalten nachweisbar sein, reicht es, wenn bei der Veröffentlichung der Nachrichten $> 2/3$ der Peers ehrlich waren und eine Thresholdsignatur über den Boardinhalt erzeugt haben.

Totale Ordnung Es gibt eine totale Ordnung (Reihenfolge) der veröffentlichten Nachrichten, diese Ordnung ist aus den veröffentlichten Daten ableitbar (z. B. durch Zeitstempel). Falls es Receipts gibt, soll die für die Ordnung nötige Information (z. B. Zeitstempel, Sequenznummer) Teil des Receipts sein. Sollte das Bulletin Board diese Ordnung ändern, muss es möglich sein, ihm dies nachzuweisen[3].

Zensurfreiheit Das Bulletin Board zensiert keine Nachrichten, d. h. es nimmt alle korrekten Nachrichten an und veröffentlicht diese. Die Einschränkung auf „korrekte" Nachrichten ist notwendig, weil das Bulletin Board legitime Gründe dafür haben kann, eine Nachricht nicht anzunehmen – z. B. wenn die Nachricht eine ungültige Signatur aufweist.

Korrektheit Alle veröffentlichten Nachrichten wurden tatsächlich an das Bulletin Board gesendet. Das bedeutet: das Bulletin Board darf keine Nachrichten „erfinden".

Für die bisher vorgestellten häufiger geforderten Eigenschaften wird in den folgenden Abschnitten bei jeder betrachteten Arbeit angegeben, ob sie gefordert werden oder nicht. Speziellere Eigenschaften sind jeweils nur dort angegeben, wo sie auch gefordert werden.

4.1.2 Veröffentlichungen speziell zu Bulletin Boards

Als erste Gruppe werden hier jene Arbeiten betrachtet, die sich explizit mit Bulletin Boards auseinandersetzen – in Abgrenzung zu den später betrachteten Artikeln, die zwar ein Bulletin Board beschreiben, dieses aber „nur" als Baustein verwenden.

Peters Eine der ersten Arbeiten, die sich mit einem Bulletin Board an sich beschäftigt, stammt von Peters [91], er diskutiert die Vor- und Nachteile unterschiedlicher Multicast-Protokolle[4] zur Implementierung eines verteilten Bulletin Boards. Darin definiert er ein Bulletin Board abstrakt als einen „public broadcast channel with memory", diese Formulierung taucht zum ersten Mal bei Cramer et al. [27] auf und wird häufig verwendet, um ein Bulletin Board zu beschreiben (z. B. in [28, 30, 53]). Konkret fordert Peters, dass ein Bulletin Board die in Tabelle 4.1 ausgeführten Eigenschaften haben sollte.

[3] Da die für die Ordnung nötigen Informationen mit veröffentlicht werden müssen, kann hier ggf. dieselbe „Nachweistechnik" verwendet werden wie für Append-Only.

[4] Ein Multicast-Protokoll dient dazu, Nachrichten an alle Mitglieder einer Gruppe von Peers zuzustellen. Der Unterschied zu „der Absender schickt die Nachricht einzeln an jedes Gruppenmitglied" besteht darin, dass Multicast-Protokolle in der Regel bestimmte Garantien machen, z. B., dass alle Gruppenmitglieder die Nachrichten in derselben Reihenfolge empfangen (sog. *Atomic Multicast*).

Eigenschaft	Gefordert/ betrachtet?	Anmerkungen
Append-Only	Ja	
Receipts	Ja	
Consistent View	Ja	Nicht explizit, folgt aber aus Peters' Forderungen *Broadcast, Agreement* und *Memory*.
Totale Ordnung	Nein	
Zensurfreiheit	Ja	
Korrektheit	Nein	
Performance†	Ja	Nicht explizit, Peters wählt Rampart aber u. a. wegen dessen Performance aus.
Verfügbarkeit‡	Ja	

†: Das Bulletin Board sollte Nachrichten möglichst schnell verarbeiten können.

‡: Jeder (berechtigte) Nutzer erhält in endlicher Zeit Antworten vom Bulletin Board.

Tabelle 4.1: Von Peters geforderte Eigenschaften eines Bulletin Boards

Heather und Lundin Einen formaleren und mehr auf das Bulletin Board als solches ausgerichteten Ansatz verfolgen Heather und Lundin [55]. Sie identifizieren die Notwendigkeit für das Bulletin Board, die Herkunft aller veröffentlichten Nachrichten nachweisen zu können. Dabei gehen sie sogar noch einen Schritt weiter und fordern, dass auch die genaue Position in der (hier geordneten) Liste von Nachrichten sowie der Zeitstempel, mit dem die Nachricht veröffentlicht wird, Teil dieses Nachweises sind. Damit werden z. B. eVoting-Systeme unterstützt, bei denen ein Wähler mehrere Stimmen abgeben kann, aber nur die Letzte gezählt wird[5]. Die Anforderungen, die Heather und Lundin an ein Bulletin Board stellen, sind in Tabelle 4.2 zusammengefasst.

Hauser und Haenni Gegenüber den Arbeiten von Peters bzw. Heather und Lundin diskutieren Hauser und Haenni [53] die gewünschten Eigenschaften eines Bulletin Boards aus Sicht der Benutzer: Sie stellen sich die Frage, wie die Schnittstelle eines möglichst generischen Bulletin Boards aussehen sollte. Dabei formulieren sie eine große Bandbreite von möglichen Herausforderungen und Anforderungen, sowohl technischer als auch organisatorischer Natur – letztere werden hier aber nicht weiter behandelt. Da Hauser und Haenni außer Append-Only nur optionale Anforderungen formulieren, wird in Tabelle 4.3 nicht zwischen optionalen und unbedingt notwendigen Eigenschaften unterschieden.

Culnane und Schneider Einen weiteren konkreten Entwurf für ein Bulletin Board liefern Culnane und Schneider [30]. Sie schlagen ein verteiltes System aus gleichwertigen Peers vor,

[5] Diese Strategie wird häufig verwendet, um einen gewissen Grad an *Coercion Resistance* zu erreichen, d. h. zu erschweren, dass Wähler zur Wahl einer bestimmten Option gezwungen werden oder ihre Stimme verkaufen können.

Eigenschaft	Gefordert/ betrachtet?	Anmerkungen
Append-Only	Ja	
Receipts	Ja	
Consistent View	Nein	
Totale Ordnung	Ja	Durch die Absender definiert.
Zensurfreiheit	Nein	
Korrektheit	Ja	Folgt aus Herkunftsnachweis.
Performance	Ja	Indirekt: Die Autoren diskutieren Möglichkeiten, die Performance zu verbessern.
Herkunftsnachweis†	Ja	

†: Das Bulletin Board muss für jede veröffentlichte Nachricht nachweisen können, von wem sie stammt.

Tabelle 4.2: Von Heather und Lundin geforderte Eigenschaften

Eigenschaft	Gefordert/ betrachtet?	Anmerkungen
Append-Only	Ja	Hier wird unbedingte Einhaltung gefordert (nicht „nur“ Nachweisbarkeit).
Receipts	Ja	
Consistent View	Ja	
Totale Ordnung	Ja	
Zensurfreiheit	Nein	
Korrektheit	Nein	
Konfliktfreiheit†	Ja	
Authentifikation‡	Ja	
Fristeinhaltung*	Ja	

†: Das Bulletin Board prüft beim Eintreffen einer Nachricht, ob diese im Konflikt zu bereits veröffentlichten Nachrichten steht und verweigert ggf. die Annahme.

‡: Die Absender von Nachrichten müssen sich gegenüber dem Bulletin Board authentifizieren, bevor sie eine Nachricht senden können. Dabei genügt es ggf., wenn der Absender nachweist, dass er zu einer bestimmten Gruppe (z. B. Wahlberechtigte) gehört.

∗: Das Bulletin Board braucht eine Strategie, um auf Nachrichten zu reagieren, die zu früh oder zu spät – z. B. bezüglich einer Wahlperiode – eintreffen.

Tabelle 4.3: Geforderte Eigenschaften bei Hauser und Haenni

bei dem die Peers sich in regelmäßigen Abständen auf einen Bulletin Board-Zustand „einigen“ und diesen dann veröffentlichen. Sie identifizieren dabei die in Tabelle 4.4 aufgeführten Anforderungen. In einer späteren Arbeit [29], in der die gleichen Autoren (zusammen mit weiteren) ein eVoting-System namens *vVote* vorstellen, wiederholen sie diese Forderungen nochmals.

Eigenschaft	Gefordert/ betrachtet?	Anmerkungen
Append-Only	Ja	
Receipts	Ja	
Consistent View	Nein	
Totale Ordnung	Nein	
Zensurfreiheit	Ja	
Korrektheit	Ja	
Konfliktfreiheit	Ja	Nur Culnane und Schneider, bei Kuldmaa nachträgliche „Filterung“.
Performance	Ja	
Authentifikation	Ja	Nur Kuldmaa.
Verfügbarkeit	Ja	Nur Kuldmaa, implizit in *Confirmable Liveness*.
Confirmable Liveness†	Ja	Nur Kuldmaa.

†: Jeder ehrliche Absender erhält in endlicher Zeit ein Receipt für seine Nachricht.

Tabelle 4.4: Anforderungen nach Culnane und Schneider / Kuldmaa

Kuldmaa Ausgehend von der Arbeit von Culnane und Schneider hat Kuldmaa [70] zunächst ein formales Gerüst zur Untersuchung von Bulletin Boards entwickelt und anschließend das Bulletin Board von Culnane und Schneider darin untersucht. Die geforderten Eigenschaften sind bis auf drei Änderungen identisch mit denen von Culnane und Schneider: Erstens wird zusätzlich zu den anderen Eigenschaften noch *Confirmable Liveness* gefordert. Confirmable Liveness bedeutet, dass jeder ehrliche Nutzer, der eine Nachricht an das Bulletin Board sendet, in endlicher Zeit ein Receipt für diese Nachricht erhält. Da sich *Confirmable Liveness* und Konfliktfreiheit gegenseitig ausschließen, wird außerdem zweitens die Forderung nach Konfliktfreiheit fallen gelassen: Es dürfen sich gegenseitig widersprechende Nachrichten veröffentlicht werden, es gibt aber zusätzlich eine (öffentliche) Filterfunktion, die bei Konflikten „entscheidet“, welche der Nachrichten gültig ist. Drittens wird bei Kuldmaa explizit gefordert, dass die Absender von Nachrichten sich authentifizieren müssen. Einen Überblick über die Forderungen gibt die Tabelle 4.4.

4.1.3 Benutzer von Bulletin Boards

In diesem Unterabschnitt werden Arbeiten betrachtet, in denen ein Bulletin Board in irgendeiner Weise benutzt, aber nicht direkt entworfen wird. Die hier herangezogenen Artikel entstammen alle der eVoting-Literatur. Wie im vorigen Unterabschnitt geht es hier hauptsächlich darum, gewünschte bzw. notwendige Eigenschaften eines Bulletin Boards zu identifizieren. Ausgewählt wurden die Veröffentlichungen danach, ob sie ausreichend genau beschreiben, was jeweils von einem Bulletin Board erwartet wird.

Beispiele für Beschreibungen, die nicht genau genug waren, sind: „[...] a piece of universally accessible memory to which all players have appendive-write access“ [62], „[...] publicly readable, and which every participant can write to [...], but nobody can delete from“ [57], „[...] publicly shared piece of memory to which all participants have read access and appendive, sequential write access [...]“ [16] sowie viele weitere Arbeiten aus dem Bereich eVoting ([2, 4, 8, 9, 11, 24, 61, 64, 87, 96, 109], um nur einige zu nennen). Gemeinsam haben alle diese Arbeiten im Bezug auf Bulletin Boards, dass irgendwie die Append-Only-Eigenschaft gefordert wird.

D-DEMOS Im Rahmen ihres eVoting-Systems „D-DEMOS“ haben Chondros et al. [25] ein einfaches, verteiltes Bulletin Board entwickelt, dessen Besonderheit darin besteht, dass die einzelnen Peers des Bulletin Boards zu keinem Zeitpunkt miteinander kommunizieren müssen. Das wird erreicht, indem mit den sogenannten *Vote Collectors* eine Instanz zwischen den Wählern und dem Bulletin Board steht. Diese Vote Collectors erstellen Receipts und stellen im Prinzip ein temporäres Bulletin Board dar. Dabei untersuchen die Autoren auch die Performance des Systems. Dass die Vote Collectors dabei der entscheidende „Flaschenhals“ sind, ist für diese Arbeit insofern interessant, als dass die Vote Collectors für die Einhaltung der (wichtigen) Eigenschaften Append-Only und Receipts zuständig sind. Die in Tabelle 4.5 dargestellten Eigenschaften, die Chondros et al. fordern, beziehen sich daher auf das Zusammenspiel von Vote Collectors und Bulletin Board.

Eigenschaft	Gefordert/ betrachtet?	Anmerkungen
Append-Only	Ja	
Receipts	Ja	Werden von den Vote Collectors ausgestellt.
Consistent View	Ja	Erst nach Ende des Wahlgangs.
Totale Ordnung	Nein	
Zensurfreiheit	Nein	
Korrektheit	Nein	
Performance	Ja	
Authentifikation	Ja	

Tabelle 4.5: Für das D-DEMOS-Bulletin Board geforderte Eigenschaften

Cramer et al. In einem der frühen Aufsätze, die explizit ein verteiltes Bulletin Board vorschlagen, präsentieren Cramer et al. [27] ein eVoting-System mit der interessanten Eigenschaft, dass der Aufwand seitens des Wählers unabhängig von der Anzahl der auswertenden Entitäten ist. In einem kurzen Absatz wird dabei (ohne weitere Details zu formulieren) vorgeschlagen, das Bulletin Board als eine Menge von Servern zu implementieren, die sich mittels eines *Byzantine Agreement*-Protokolls [74] untereinander synchronisieren. An das Bulletin Board werden dabei die in Tabelle 4.6 aufgelisteten Anforderungen gestellt.

Eigenschaft	Gefordert/ betrachtet?	Anmerkungen
Append-Only	Ja	
Receipts	Nein	
Consistent View	Ja	Implizit: Jeder soll alle Kommunikation mit dem Bulletin Board lesen können.
Totale Ordnung	Nein	
Zensurfreiheit	Ja	
Korrektheit	Nein	
Verfügbarkeit	Ja	Konkret wird gefordert, dass Denial-of-Service-Angriffe unmöglich sein sollen.
Authentifikation	Ja	Implizit: Jede Partei soll ihre eigene Board-Partition erhalten.

Tabelle 4.6: Forderungen an ein Bulletin Board bei Cramer et al.

Bulletin Board für faire Multiparty Computation Einen Verwendungszweck für Bulletin Boards außerhalb von eVoting-Protokollen zeigen Choudhuri et al. in [26] auf: Sie verwenden ein Bulletin Board, um in *Multiparty Computations perfekte Fairness* zu erreichen. Multiparty Computations sind Protokolle, in denen mehrere Parteien gemeinsam eine Funktion berechnen, die jeweiligen Eingaben sollen dabei geheim bleiben. Perfekte Fairness bedeutet hier, dass entweder alle Beteiligten ihren Teil des Funktionsergebnisses erhalten oder keiner der Beteiligten. Interessant ist an der Arbeit von Choudhuri et al. außerdem, dass die Autoren eine Bulletin Board-Implementierung auf Basis einer Blockchain vorschlagen und ein entsprechendes Beispiel mit Bitcoin implementieren. Unabhängig von der konkreten Implementierung erwarten die Autoren die in Tabelle 4.7 aufgeführten Eigenschaften von einem Bulletin Board.

Adida (Helios) In seiner Dissertation [1] beschäftigt sich Ben Adida – der Entwickler des oft referenzierten *Helios*-Wahlsystems – ausführlich mit kryptographischen Wahlsystemen. Dabei hält er fest, dass „kryptographische Wahlsysteme um ein zentrales, digitales Bulletin Board herum“ gebaut sind[6]. Als Anforderungen an dieses Bulletin Board identifiziert er die in Tabelle 4.8 gelisteten Eigenschaften.

[6]Im Original „Cryptographic voting protocols revolve around a central, digital bulletin board.“ [1, S. 34]

Eigenschaft	Gefordert/ betrachtet?	Anmerkungen
Append-Only	Ja	
Receipts	Ja	Stärkere Forderung: *Beweis*, dass Nachricht veröffentlicht wurde.
Consistent View	Ja	
Totale Ordnung	Nein	
Zensurfreiheit	Ja	
Korrektheit	Nein	

Tabelle 4.7: Von Choudhuri et al. geforderte Eigenschaften

Eigenschaft	Gefordert/ betrachtet?	Anmerkungen
Append-Only	Ja	
Receipts	Nein	
Consistent View	Ja	Adida schlägt vor, dass Beobachter Consistent View sicherstellen sollen.
Totale Ordnung	Ja	Aus Eingangszeitpunkt beim Bulletin Board.
Zensurfreiheit	Nein	
Korrektheit	Nein	
Authentifikation	Ja	

Tabelle 4.8: Von Adida geforderte Eigenschaften

Araújo et al. Auch Araújo et al. entwickeln in [5] ein eVoting-System und auch dieses System baut auf einem zentralen Bulletin Board auf, an das Araújo et al. die in Tabelle 4.9 dargestellten Forderungen stellen.

Tjøstheim et al. In einer Arbeit aus 2010 stellen Tjøstheim et al. [102] einen Analyserahmen für Wahlsysteme vor, mit dem unter anderem auch eVoting-Systeme systematisch auf bestimmte Eigenschaften wie Geheimhaltung oder Integrität hin untersucht werden können. In diesem Kontext werden Bulletin Boards natürlich nur abstrakt behandelt, Tjøstheim et al. formulieren daher nur zwei zentrale Forderungen: Append-Only und *verifizierbares Posten*. Letzteres ist im Prinzip eine abgeschwächte Form von Receipts, allerdings fordern Tjøstheim et al. nicht, dass der Wähler im Zweifel auch *nachweisen* kann, dass seine Nachricht nicht oder nicht korrekt veröffentlicht wurde.

Eigenschaft	Gefordert/ betrachtet?	Anmerkungen
Append-Only	Ja	
Receipts	Nein	
Consistent View	Ja	Formulierung wie bei Cramer (Tabelle 4.6)
Totale Ordnung	Ja	
Zensurfreiheit	Nein	
Korrektheit	Nein	
Anonymes Posten†	Ja	

†: Die Stimmabgabe der Wähler auf dem Bulletin Board muss über einen anonymen Kanal erfolgen.

Tabelle 4.9: Für das eVoting-System von Araújo et al. notwendige Eigenschaften des Bulletin Boards

4.1.4 Übersicht und Einordnung der Anforderungen

In den letzten beiden Unterabschnitten wurden mehrere Arbeiten vorgestellt, in denen Bulletin Boards entwickelt bzw. benutzt (und beschrieben) wurden. In diesem Unterabschnitt werden die von den jeweiligen Autoren identifizierten Anforderungen an Bulletin Boards zusammengefasst und gewichtet. Die Reihenfolge folgt dabei Tabelle 4.10, die zugleich einen schnellen Überblick über die Anforderungen bietet.

Append-Only Alle betrachteten Arbeiten fordern, dass Nachrichten, die einmal vom Bulletin Board veröffentlicht wurden, nicht wieder entfernt oder verändert werden können. Sollte es doch dazu kommen, dann soll es zumindest möglich sein, dem Bulletin Board dieses Fehlverhalten nachzuweisen. Diese Einschränkung ist in der Praxis nicht nur deswegen nötig, weil sie unter schwächeren Annahmen erfüllt werden kann (vgl. Definition auf Seite 36): Es ist nämlich unmöglich, Daten „unlöschbar“ zu speichern; selbst wenn das System keine Löschvorgänge erlaubt, können immer die eigentlichen Datenträger zerstört werden. Da alle untersuchten Arbeiten die Append-Only-Eigenschaft fordern, darf sie als zentral für ein Bulletin Board angesehen werden.

Receipts Auch Receipts werden in fast allen betrachteten Arbeiten gefordert und gehören folglich zu den zentralen Anforderungen an ein Bulletin Board. Wo sie nicht explizit gefordert werden, wird meist ohnehin von einem korrekt arbeitenden Bulletin Board ausgegangen, d. h. Receipts hätten keinen Nutzen.

	Peters	Heather	Hauser	Culnane	Kuldmaa	Chondros	Cramer	Choudhuri	Adida	Araújo	Tjøstheim
Append-Only	Ja	Ja	Ja*	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Receipts	Ja	Ja	Ja	Ja	Ja	Ja	Nein	Ja*	Nein	Nein	Ja*
Consistent View	Ja	Nein	Ja	Nein	Nein	Ja*	Ja*	Ja	Ja	Ja*	Nein
Totale Ordnung	Nein	Ja	Ja	Nein	Nein	Nein	Nein	Nein	Ja	Ja	Nein
Zensurfreiheit	Ja	Nein	Nein	Ja	Ja	Nein	Ja	Ja	Nein	Nein	Nein
Korrektheit	Nein	Ja	Nein	Ja	Ja	Nein	Nein	Nein	Nein	Nein	Nein
Authentifikation	Nein	Nein	Ja	Nein	Ja	Ja	Ja	Nein	Ja	Nein	Nein
Performance	Ja	Ja	Nein	Ja	Ja	Ja	Nein	Nein	Nein	Nein	Nein
Verfügbarkeit	Ja	Nein	Nein	Nein	Ja	Nein	Ja*	Nein	Nein	Nein	Nein
Konfliktfreiheit	Nein	Nein	Ja	Ja	Nein	Nein	Nein	Nein	Nein	Nein	Nein
Anonymes Posten	Nein	Nein	Nein	Nein	Nein	Nein	Nein	Nein	Nein	Ja	Nein
Herkunftsnachweis	Nein	Ja	Nein	Nein	Nein	Nein	Nein	Nein	Nein	Nein	Nein
Fristeinhaltung	Nein	Nein	Ja	Nein	Nein	Nein	Nein	Nein	Nein	Nein	Nein

Tabelle 4.10: Übersicht über die Anforderungen der betrachteten Arbeiten an ein Bulletin Board. Aus Platzgründen ist jeweils nur der Hauptautor angegeben. „Ja“ bedeutet hier, dass die Eigenschaft explizit gefordert oder anderweitig berücksichtigt wurde. Ein „Ja*“ bedeutet, dass sich die jeweilige Forderung der Autoren stärker von der in Unterabschnitt 4.1.1 bzw. 4.1.2 und 4.1.3 formulierten Forderung unterscheidet. Diese Unterschiede werden in Unterabschnitt 4.1.2 bzw. 4.1.3 näher beschrieben. Die abgesetzten Eigenschaften wurden nur selten gefordert oder erwähnt, außerdem fehlt hier *Confirmable Liveness* von Kuldmaa, weil diese ohnehin aus Zensurfreiheit, Receipts und Verfügbarkeit folgt.

Consistent View Ein Consistent View für alle lesenden Parteien gehört ebenfalls zu den häufiger formulierten Forderungen. In den Arbeiten, in denen ein Consistent View nicht explizit gefordert wird, ist ein solcher wahrscheinlich implizit in „Append-Only“ mit gemeint – denn die Forderung nach Append-Only hat ohne die Forderung nach einem Consistent View kaum praktischen Nutzen: Das Bulletin Board könnte jedem Teilnehmer nur die von diesem Teilnehmer geschriebenen Nachrichten anzeigen – was die Idee einer öffentlichen Datenstruktur ad absurdum führt. Dementsprechend ist auch Consistent View eine der zentralen Eigenschaften eines Bulletin Boards.

Totale Ordnung Die Forderung nach einer eindeutigen Reihenfolge der Nachrichten ist häufig durch die Anwendung in einem eVoting-System motiviert, welches Coercion Resistance erreichen möchte, indem die Wähler mehrfach abstimmen können, wobei nur die letzte Stimme tatsächlich gewertet wird [5, 53]. Eine wichtige Frage ist dabei aber auch, wie die Ordnung definiert wird – während z. B. Araújo et al. und Adida den Eingangszeitpunkt beim Bulletin Board als Grundlage nehmen, definieren bei Heather und Lundin die Absender die Reihenfolge der Nachrichten. Weil eine totale Ordnung der Nachrichten auch für andere Anwendungen wie z. B. Auktionen notwendig ist und sonstigen Anwendungen nicht schadet, sollte ein allgemeines Bulletin Board wenn möglich zumindest die Ordnung anhand des Eingangszeitpunkts unterstützen, indem jede Nachricht mit einem Eingangs-Zeitstempel versehen wird.

Bei einem verteilten Bulletin Board ist ein solcher Eingangs-Zeitstempel aber nicht ohne weiteres machbar: Wenn die Absender einer Nachricht ihre Nachricht an einen beliebigen Peer bzw. in beliebiger Reihenfolge an die Peers senden können, dann kann es bei parallel eintreffenden Nachrichten Uneinigkeit der Peers darüber geben, welche Nachricht zuerst eingetroffen ist. Wird die parallele Verarbeitung von Nachrichten verhindert (oder optimistisch: erkannt und dann beide abgelehnt), ist eine totale Ordnung möglich – allerdings zu einem hohen Preis bezüglich der Performance.

Zensurfreiheit Die Forderung nach Zensurfreiheit gehört – ähnlich wie Receipts und Consistent View – zu denen, die zwar nicht immer explizit genannt, sehr wahrscheinlich aber implizit in Formulierungen wie „[...] allows various parties [...] to publish information“ [55, S. 242 (Heather/Lundin)] gemeint sind[7]. Würde man dem Bulletin Board nämlich erlauben, Nachrichten beliebig zu zensieren, so wäre es dem Sinn nach gerade kein *öffentliches* Board mehr. Dementsprechend ist auch Zensurfreiheit eine zentrale Forderung an ein Bulletin Board.

Korrektheit, Herkunftsnachweis und Authentifikation Diese Eigenschaften werden gemeinsam betrachtet, weil sie eng verwandt sind: In allen drei Fällen muss das Bulletin Board für die *nachweisliche* Erfüllung pro veröffentlichter Nachricht einen Beweis dafür liefern, dass

[7] Weitere entsprechende Formulierungen: „Its purpose is to allow the parties involved in a protocol to publish messages [...]“ [53, S. 242 (Hauser/Haenni)]; „[...] bulletin board performs as a public broadcast channel [...]“ [5, S. 334 (Araújo et al.)].

diese Nachricht von einem Mitglied einer bestimmten Gruppe versendet wurde. Für Korrektheit besteht diese Gruppe aus allen Parteien außer dem Bulletin Board selbst, für Herkunftsnachweise enthält die Gruppe nur den tatsächlichen Absender und für Authentifikation alle „Schreibberechtigten". Daran ist auch ein wichtiger, wenngleich subtiler Unterschied zwischen Herkunftsnachweis und den beiden anderen Eigenschaften erkennbar: Für Korrektheit und Authentifikation ist es nicht unbedingt notwendig, dass das Bulletin Board die genaue Identität des Absenders kennt[8].

Obwohl in der Mehrheit der untersuchten Veröffentlichungen mindestens eine der drei Eigenschaften gefordert wird, werden sie hier nicht als notwendige Eigenschaften behandelt. Der Grund dafür ist die in Abschnitt 2.5 erläuterte Abgrenzung zwischen Anwendung und Bulletin Board: Wenn die Anwendung z. B. einen Herkunftsnachweis erfordert, dann kann das seitens der Anwendung recht einfach dadurch realisiert werden, dass alle Nachrichten signiert werden müssen. Nachrichten mit ungültiger oder unbekannter Signatur können dann zwar trotzdem auf dem Bulletin Board erscheinen, werden aber im weiteren Verlauf von der Anwendung ignoriert (z. B. bei der Stimmauszählung nicht berücksichtigt).

Performance Bei dieser Eigenschaft stellt sich zunächst die Frage, was damit überhaupt gemeint ist, da es mehrere Aspekte zu berücksichtigen gilt: Der offensichtlichste Aspekt ist die Anzahl an Nachrichten/Daten, die das Bulletin Board selbst pro Zeiteinheit annehmen kann[9]. Weniger offensichtlich ist der notwendige Aufwand bei den Absendern von Nachrichten und Lesern des Bulletin Boards: Müssen sie z. B. digitale Signaturen erstellen bzw. überprüfen? Wenn ja, wie viele (in Abhängigkeit von der Anzahl der Nachrichten und beteiligten Parteien)? Und wie viel Kommunikation ist neben dem reinen Versenden der Nachricht notwendig? Für einen praktischen Einsatz sind diese Performance-Eigenschaften ähnlich wichtig wie die Sicherheitseigenschaften und sollten daher beim Entwurf eines Bulletin Boards nicht außer Acht gelassen werden.

Verfügbarkeit Explizit wird Verfügbarkeit nur von drei der untersuchten Arbeiten gefordert, es ist aber offensichtlich, dass das Bulletin Board verfügbar sein muss, um irgendeinen Nutzen zu haben. Insbesondere für die Anwendung in einem eVoting-System für größere Wahlen würde schon ein kurzer „Aussetzer" von wenigen Sekunden zu Tausenden erfolglosen Versuchen für eine Stimmabgabe führen – das wäre schon ohne Berücksichtigung juristischer Probleme nicht akzeptabel. Damit ist eine hohe Verfügbarkeit eine zentrale Forderung an ein Bulletin Board. Absolute Verfügbarkeit ist in der Praxis aber nicht erfüllbar – Hardwareausfälle sind immer möglich. Darum wird Verfügbarkeit im weiteren Verlauf wie folgt betrachtet: Das Bulletin Board darf keinen „Single Point of Failure" haben und die erwartete Verfügbarkeit muss mit steigender Anzahl an Peers ebenfalls steigen.

[8] Das kann z. B. durch Verwendung von Ring- oder Gruppensignaturen erreicht werden ([95] bzw. [23]).

[9] Ein Beispiel für die nötige Größenordnung: In vielen eVoting-Systemen sendet jeder Wähler eine Nachricht mit seiner Stimmabgabe an das Bulletin Board. Im Fall einer deutschen Bundestagswahl bedeutet das innerhalb einer Wahlperiode von zehn Stunden, dass *im Mittel* ca. 1.700 Nachrichten pro Sekunde verarbeitet werden müssten.

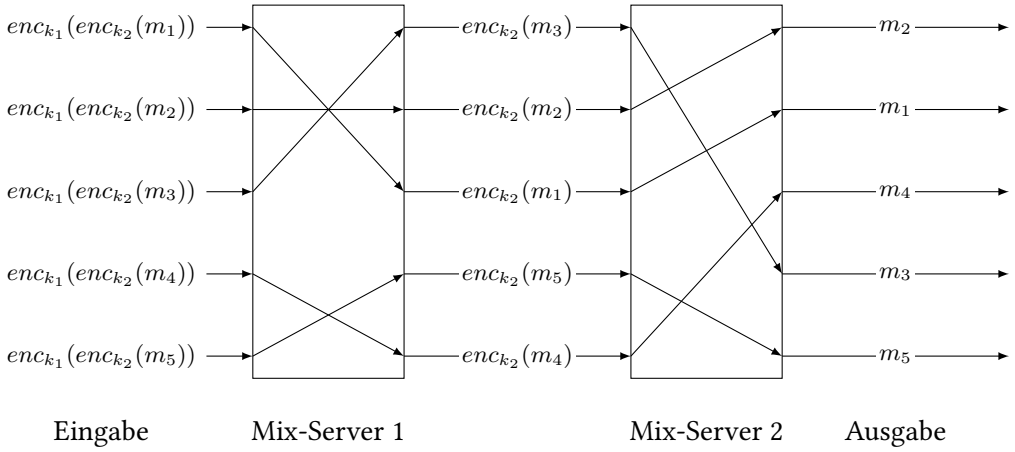


Abbildung 4.1: Einfaches Mixer-Netzwerk mit zwei Mixern, die jeweils die Eingabe zufällig permutieren und eine „Verschlüsselungsschicht“ entfernen.

Konfliktfreiheit Diese Eigenschaft wird hier nicht weiter berücksichtigt. Das liegt zum einen daran, dass Konfliktfreiheit nur in zwei der untersuchten Arbeiten gefordert wird. Zum anderen und vor allem aber gilt hier eine ähnliche Argumentation wie für Korrektheit, Herkunftsnachweis und Authentifikation: Das Verständnis davon, was „sich widersprechende“ Nachrichten sind, hängt stark von der Anwendung ab, dementsprechend kann ein allgemeines Bulletin Board diese Eigenschaft gar nicht erfüllen. Da zudem die „Regeln“ zur Erkennung sich widersprechender Nachrichten ohnehin öffentlich sein müssen (sonst könnte das Bulletin Board willkürlich zensieren), kann diese „Filterung“ auch durch die Leser vorgenommen werden – so z. B. von Kuldmaa [70] vorgeschlagen.

Anonymes Posten Araújo et al. fordern (als einzige) einen anonymen Kanal zum Bulletin Board. Mit Blick auf die anderen untersuchten Arbeiten, die ohne einen solchen auskommen, ist bereits festzuhalten, dass die Möglichkeit zum anonymen Posten sicher keine zentrale Forderung an ein Bulletin Board darstellt. Praktisch umgesetzt werden solche anonymen Kanäle meist durch ein Netzwerk von „Mix-Servern“. Ein solches Netzwerk mit zwei Mixern ist in Abbildung 4.1 gezeigt[10]: Die Mixer nehmen jeweils mehrere (mehrfach) verschlüsselte Nachrichten an, mischen diese, entfernen eine „Schicht“ der Verschlüsselung und leiten das Ergebnis an den nächsten Mixer weiter. So lange dabei mindestens einer der Mixer seine Permutation geheim hält, bleibt auch geheim, welche Eingabe zu welcher Ausgabe gehört[11]. Ein solches Netzwerk könnte prinzipiell auch Teil eines verteilten Bulletin Boards sein, da die Details aber für das Bulletin Board an sich irrelevant sind, wird die Forderung nach einem anonymen Kanal hier nicht weiter verfolgt.

[10] Die Darstellung ist bewusst sehr oberflächlich gehalten, um das Konzept deutlich zu machen – für mehr Details sei auf die entsprechende Literatur verwiesen: [22, 86, 90], um nur einige zu nennen.

[11] Tatsächlich gibt es hier noch einige weitere Probleme zu berücksichtigen, z. B. könnte die Länge der Ein- und Ausgaben eine Korrelation erlauben – das geht aber über den Rahmen dieser Arbeit hinaus.

Fristeinhaltung Auch diese Forderung wird nur in einer der untersuchten Arbeiten gestellt und analog zur Konfliktfreiheit kann auch Fristeinhaltung durch die Anwendung „nachträglich“ geprüft werden. Anders als bei der Konfliktfreiheit muss hier aber das Bulletin Board mitwirken, indem es alle Nachrichten mit einem Eingangs-Zeitstempel versieht. Folglich wird im weiteren Verlauf dieser Arbeit zwar die Fristeinhaltung als Anforderung nicht weiter berücksichtigt, es wird aber diskutiert, wie das Bulletin Board die notwendigen Zeitstempel ausstellen kann.

4.1.5 Zusammenfassung

Die in den letzten Unterabschnitten beschriebenen Eigenschaften sind dort – wie eingangs erklärt – bewusst idealisiert formuliert. In einer praktischen Umsetzung können sie daher nicht oder nur unter sehr starken Annahmen erfüllt werden.

Dennoch sollte ein sicheres Bulletin Board mindestens die Eigenschaften Append-Only, Receipts, Verfügbarkeit und Consistent View unter *realistischen* Annahmen – etwa dass in einem verteilten System nur eine Minderheit der Peers kompromittiert ist – erfüllen. Küsters et al. fassen dies wie folgt zusammen:

> "The key property required is that every party has access to the bulletin board and that it provides the same view to everybody." (Küsters et al., 2016 in [73, S. 23])

Zusätzlich muss für eine reale Anwendung aber auch die Effizienz berücksichtigt werden – die Absender von Nachrichten, Leser und auch das Bulletin Board selbst sowie möglicherweise andere beteiligte Parteien sollten pro veröffentlichter Nachricht möglichst wenig Ressourcen beanspruchen.

Zu den Problemen bei der Implementierung eines Bulletin Boards finden sich auch in der Literatur entsprechende Bemerkungen:

> "In practice, implementing such a bulletin board is one of the more challenging engineering aspects of cryptographic voting, as one must worry about availability issues beyond data corruption, such as denial-of-service attacks for both data publication and access." (Adida, 2006 in [1, S. 35])

> "[...] most E2E schemes presuppose the existence of a public append-only bulletin board. Implementing this, however, proved to be a major technical challenge, which invariably leads to a relaxation of security properties." (Zagórski et al., 2013 in [109, S. 11])

> "Achieving these properties in an implementation is not so straightforward. A current view is that ’we don’t know how to build a secure bulletin board’ [...]" (Culnane und Schneider, 2014 in [30, S. 169])

Die Anforderungen sind also breit gefächert und können nicht ohne Weiteres in eine Implementierung umgesetzt werden.

4.2 Angriffsszenarien und Gegenmaßnahmen

Nachdem im letzten Abschnitt die Anforderungen an ein Bulletin Board identifiziert wurden, werden in diesem Abschnitt verschiedene Angriffsszenarien rund um Bulletin Boards diskutiert. Der Fokus liegt dabei auf den als zentral ausgemachten Eigenschaften. Dabei wird unterschieden zwischen Angriffen von außen und Angriffen „von innen“, bei denen das Bulletin Board selbst bzw. sein Betreiber der Angreifer ist. In den folgenden Unterabschnitten werden verschiedene Angriffe kurz beschrieben und mögliche/übliche Gegenmaßnahmen behandelt.

4.2.1 Angriffe von außen auf das Bulletin Board

Bei Angriffen von außen bleibt das Bulletin Board selbst ehrlich, d. h. der Angreifer kann aus Sicht des Bulletin Boards „nur“ die Kommunikation mit dem Bulletin Board beeinflussen. Das schließt das Verzögern, freie Erfinden, Verändern usw. von Nachrichten mit ein (falls das Bulletin Board als verteiltes System implementiert ist, gilt das auch für Nachrichten zwischen den Peers), außerdem darf der Angreifer beliebig viele der Absender und Leser kompromittieren.

Zu Angriffen von außen finden sich in der Literatur einige wenige Anmerkungen: Adida [1, 4] nennt Zensurversuche und das Injizieren falscher Daten (im Sinne einer Verletzung von Append-Only) als mögliche Ziele eines Angreifers – da das Bulletin Board in der aktuellen Betrachtung jedoch nicht kompromittiert ist und eine nachträgliche Änderung der Boardinhalte unter dieser Annahme leicht verhindert werden kann, wird hier nur die Möglichkeit der Zensur betrachtet.

A.1 (Zensurfreiheit) *Der Angreifer kann von ihm unerwünschte Nachrichten auf dem Übertragungsweg verschwinden lassen bzw. bei einem verteilten Bulletin Board den Veröffentlichungsprozess stören.*

Prinzipiell kann diesem Angriff wenig entgegen gesetzt werden, es ist aber möglich, ihn zu erschweren: Die Nachrichten ans Bulletin Board können verschlüsselt übertragen werden, d. h., der Angreifer kann die Nachrichten nicht aufgrund ihres Inhalts „aussortieren“ – gerade beim Einsatz für Wahlen ist aber häufig schon die Absenderidentität ausreichend, um zu erahnen, was der Inhalt sein wird (also für wen oder was der Absender seine Stimme abgibt).

Hauser und Haenni [53] nennen noch zwei weitere Möglichkeiten für externe Angreifer:

A.2 (Verfügbarkeit/Performance) *Der Angreifer „flutet“ das Bulletin Board mit Nachrichten und sorgt so dafür, dass legitime Nachrichten nicht mehr veröffentlicht werden können, weil die Ressourcen (Speicher, CPU-Zeit, Netzwerkanbindung) des Bulletin Boards belegt werden.*

Auch gegen diesen Angriff kann nur wenig getan werden: Das Bulletin Board kann möglichst leistungsfähig ausgelegt sein, um einen solchen Angriff auszuhalten. Die üblichen Gegenmaßnahmen gegen Angriffe dieser Art sind hier problematisch, weil sie in der Regel bestimmte Absenderadressen sperren oder eingehende Daten „vorfiltern“ – dabei könnten auch legitime Nachrichten aussortiert werden, was im Sinne der Zensurfreiheit inakzeptabel ist.

A.3 (Verfügbarkeit/Performance) *Der Angreifer zeichnet (legitime) Nachrichten auf und sendet sie wiederholt an das Bulletin Board.*

Neben der ggf. eingeschränkten Verfügbarkeit durch eine Vielzahl an Nachrichten könnte die Anwendung, die das Bulletin Board verwendet, mehrfache Nachrichten verbieten und ggf. den vermeintlichen Absender bestrafen (z. B. für den scheinbaren Versuch, mehrere Stimmen abzugeben).

Als Gegenmaßnahme kann das Bulletin Board von den Absendern fordern, dass sie jede Nachricht mit einer Nonce versehen – dann würde ein ehrlicher Absender nie mehrere identische Nachrichten versenden, dementsprechend kann das Bulletin Board eine Nachricht, die es bereits empfangen hat, als Duplikat ablehnen.

Weil der Angreifer das Netzwerk kontrolliert, sind noch weitere Angriffsvarianten – insbesondere gegen die Verfügbarkeit – denkbar. Ein interessanteres Szenario ist noch das folgende:

A.4 (Verfügbarkeit/Zensurfreiheit) *Der Angreifer kann die Nachrichten auf dem Übertragungsweg verändern.*

Dieser Angriff kann relativ einfach erkannt werden, falls eine Public-Key-Infrastruktur vorhanden ist: Die Absender signieren alle Nachrichten. Sollte der Angreifer dann eine Nachricht eines ehrlichen Absenders verändern, ist die Signatur ungültig und das Bulletin Board kann die Nachricht ablehnen. Das allerdings führt dazu, dass die Verfügbarkeit eingeschränkt wird – im Ergebnis ist das fast äquivalent zu Angriff A.1.

Zusammenfassend lässt sich also sagen, dass die Angriffsfläche für Angriffe von außen durch Signaturen, Erkennung von Duplikaten und ein möglichst leistungsfähiges Bulletin Board reduziert werden kann, ganz verhindern kann man Angriffe im hier verwendeten Angreifermodell jedoch grundsätzlich nicht. Denn der Angreifer ist immer in der Lage, einzelne Absender oder auch das Bulletin Board selbst komplett von der Kommunikation auszuschließen.

4.2.2 Angriffe durch das Bulletin Board selbst

Im Gegensatz zum letzten Unterabschnitt wird hier davon ausgegangen, dass das Bulletin Board selbst kompromittiert ist. Das bedeutet insbesondere, dass Inhalte auf dem Bulletin Board durch den Angreifer beliebig verändert werden können, dass das Bulletin Board jedem Leser andere Daten liefern kann usw. Daraus folgt direkt, dass alle oben genannten Eigenschaften verletzt werden können – folglich zielen fast alle Gegenmaßnahmen darauf ab, solche Verletzungen erkennbar und nachweisbar zu machen.

Eine ganz allgemeine Gegenmaßnahme ist die Verteilung der Funktionalität auf mehrere Peers, d. h. ein verteiltes Bulletin Board. Noch vor – je nach Design – höherer Performance und besserer Verfügbarkeit ist das in der Regel der Grund dafür, dass ein Bulletin Board als verteiltes System ausgeführt wird: Solange nicht mehr als ein bestimmter Anteil der Peers kompromittiert wird, bleiben alle gewünschten Eigenschaften erfüllt[12]. Aber auch bei einem verteilten Bulletin Board ist denkbar, dass ein Angreifer das Bulletin Board übernimmt, indem er einfach „genügend“ Peers

[12] Oft werden dafür Byzantine Agreement-Protokolle vorgeschlagen, diese „funktionieren“ in der Regel, solange weniger als ein Drittel der Peers kompromittiert ist.

kompromittiert. Bei der folgenden Beschreibung von Angriffen wird daher angenommen, dass der Angreifer beliebig viele Peers kontrollieren kann. Außerdem können auch hier Absender und Leser kompromittiert sein.

Da die Angriffe an sich trivial sind – weil das Bulletin Board selbst der Angreifer ist, gibt es nichts, was einen Angriff verhindern könnte – ist hier vor allem interessant, wie die Überprüfbarkeit durch Dritte implementiert wird. Wenn bei den Gegenmaßnahmen digitale Signaturen genannt werden, ist für verteilte Bulletin Boards jeweils eine Thresholdsignatur gemeint, für deren Erzeugung der gleiche Anteil an Peers benötigt wird, wie er auch für korrekte Funktionalität des Bulletin Boards gebraucht wird.

A.5 (Append-Only) *Das Bulletin Board kann Nachrichten nach Veröffentlichung verändern oder entfernen.*

Eine solche Veränderung kann erkannt werden, indem regelmäßig der Inhalt des Bulletin Boards gelesen und mit früheren Antworten des Bulletin Boards verglichen wird. Damit eine solche Veränderung auch nachgewiesen werden kann, sollte das Bulletin Board seine Antworten jeweils digital signieren. Falls dann Inkonsistenzen entdeckt werden, kann anhand der Signaturen bewiesen werden, dass die inkonsistenten Daten tatsächlich genau so vom Bulletin Board gesendet wurden.

A.6 (Receipts) *Das Bulletin Board kann ungültige Receipts ausstellen.*

Damit ungültige Receipts von den Absendern als ungültig erkannt werden und die Receipts überhaupt als ein Nachweis dienen können, muss sich ein Receipt direkt auf die damit bestätigte Nachricht beziehen. Das kann z. B. erreicht werden, indem das Receipt eine Signatur des Bulletin Boards über die Nachricht enthält – damit kann der Absender der Nachricht erstens prüfen, ob das Receipt tatsächlich zur abgeschickten Nachricht passt und zweitens könnte er damit später nachweisen, dass seine Nachricht zwar vom Bulletin Board akzeptiert, aber nicht veröffentlicht wurde.

A.7 (Receipts) *Das Bulletin Board kann Receipts mehrfach verwenden (Clash Attack).*

Selbst wenn das Receipt eine Signatur des Bulletin Boards enthält, kann es vorkommen, dass das Bulletin Board ein (gültiges) Receipt ausstellt und die Nachricht unbemerkt trotzdem nicht veröffentlicht – falls nämlich eine Nachricht mehrfach (ggf. von verschiedenen Absendern) auftritt[13]. Indem Absender ihren Nachrichten immer eine Nonce beifügen, kann verhindert werden, dass dieselbe Nachricht mehrfach auftritt.

A.8 (Consistent View) *Das Bulletin Board kann unterschiedlichen Lesern beliebige Teilmengen von Nachrichten anzeigen.*

Bei der Append-Only-Eigenschaft wurde bereits vorgeschlagen, dass das Bulletin Board seine Antworten grundsätzlich digital signiert. Das alleine reicht jedoch noch nicht aus, um eine Verletzung des Consistent View zu erkennen – dafür müssen die Leser die Antworten, die sie vom Bulletin Board bekommen haben, untereinander vergleichen.

Dabei darf aber nicht einfach auf Gleichheit hin verglichen werden, weil sich der Inhalt des Bulletin Boards jederzeit durch neu eintreffende Nachrichten ändern kann. Stattdessen

[13] Diese Art von Angriff wird von Küsters et al. in [69] beschrieben und dort als *Clash Attack* bezeichnet – Küsters et al. nehmen als Angreifer kompromittierte Wahlmaschinen an, das Prinzip ist aber dasselbe.

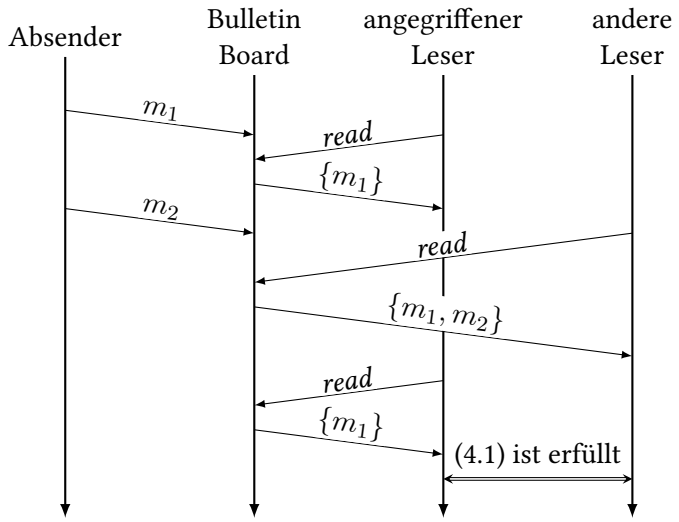


Abbildung 4.2: Möglicher Angriff durch das Bulletin Board: Dem angegriffenen Leser wird die Nachricht m_2 nie angezeigt. Falls der Leser seinen View mit dem der anderen vergleicht, fällt das zwar auf, wird aber nach (4.1) nicht als Angriffsversuch des Bulletin Boards erkannt.

muss für jede (veröffentlichte) Nachricht m und alle Paare A, A' von Antworten des Bulletin Boards an die Leser gelten:

$$m \in A \wedge m \notin A' \Rightarrow A' \subsetneq A \tag{4.1}$$

Außerdem müssen sich die Leser zunächst gegenseitig „finden“ – dabei darf natürlich nicht darauf vertraut werden, dass das Bulletin Board eine Liste mit allen Lesern führt, d. h. dieses „Finden“ muss außerhalb des Bulletin Boards ablaufen.

A.9 (Consistent View) *Das Bulletin Board kann einzelnen Lesern Nachrichten vorenthalten.*
Weil die Antworten des Bulletin Boards an die Leser alle unabhängig voneinander sind, kann das Bulletin Board wie in Abbildung 4.2 gezeigt einzelnen Lesern nur eine Teilmenge der Nachrichten anzeigen, ohne dass eine Prüfung durch die Leser mittels (4.1) fehlschlägt: Der angegriffene Leser bekommt immer nur die Nachricht m_1 zu sehen, während den anderen Lesern alle bisher eingegangenen Nachrichten angezeigt werden – analog könnte dem angegriffenen Leser auch m_1 vorenthalten und alle anderen Nachrichten angezeigt werden.
Den Lesern könnte durch Vergleich der Antworten zwar auffallen, dass diese nicht plausibel sind, sie hätten aber keinen Nachweis dafür. Damit ein solcher Angriff für die Leser nachweisbar wird, sollte das Bulletin Board seine Antworten an die Leser mit aufsteigenden Sequenznummern[14] versehen (und die Antwort inklusive der Sequenznummer signieren). Beim Vergleich der Views der Leser muss dann für alle Paare A, A' von

[14] Alternativ sind auch Zeitstempel möglich – diese haben den zusätzlichen Vorteil, dass der Leser unplausible Abweichungen recht einfach erkennen kann.

Antworten des Bulletin Boards gelten:

$$\text{seq}_A > \text{seq}_{A'} \Rightarrow A' \subseteq A \tag{4.2}$$

$$\text{seq}_A \neq \text{seq}_{A'} \tag{4.3}$$

A.10 (Totale Ordnung) *Das Bulletin Board kann falsche Zeitstempel vergeben.*[15]

Falls das Bulletin Board dafür zuständig ist, die Ordnung zu definieren, kann diese auch durch das Bulletin Board manipuliert werden, z. B. indem einzelne Nachrichten erst verzögert angenommen werden. Dabei genügt es auch nicht, wenn die Absender ihre Nachrichten mit einem Zeitstempel versehen, ein Angriff durch das Bulletin Board wäre damit Dritten gegenüber noch immer nicht nachweisbar, schließlich könnte auch der Zeitstempel des Absenders falsch sein. Dazu kommt in einem asynchronen oder teilsynchronen Netzwerk noch, dass die Uhren der einzelnen Teilnehmer auseinander driften.

Eine Lösung dafür schlagen Heather und Lundin [55] vor: Bei ihnen bestimmt allein der Absender einer Nachricht, an welcher Stelle in der Sequenz von Nachrichten eine bestimmte Nachricht auftauchen soll. Dafür fragt der Absender zunächst nach dem aktuellen Inhalt des Boards und fügt seiner Nachricht denn einen Hashwert über den Boardinhalt bei – so kann jeder Leser überprüfen, ob der Boardinhalt vor dieser Nachricht dem entspricht, was der Absender „gefordert“ hat.

A.11 (Zensurfreiheit/Verfügbarkeit) *Das Bulletin Board kann Nachrichten ignorieren, d. h. keine Receipts dafür ausstellen und sie nicht veröffentlichen.*

Dieser Angriff kann zwar durch den Absender mit hoher Wahrscheinlichkeit erkannt, allerdings nicht gegenüber Dritten nachgewiesen werden. Das liegt zum einen daran, dass der Absender sich im Allgemeinen selbst nicht sicher sein kann, ob das Bulletin Board die Nachricht ignoriert oder ob sie auf dem Weg zum Bulletin Board verloren gegangen ist. Zum anderen kann ein „Absender“ jederzeit einfach behaupten, er hätte eine Nachricht gesendet, ohne dass diese Behauptung widerlegt werden kann.

Letzteres ist auch der Grund, warum eine Konstruktion wie in Angriff A.1 – Nachrichten werden zunächst verschlüsselt ans Bulletin Board gesendet – hier nicht ohne Weiteres möglich ist: Würde das Bulletin Board den Empfang einer verschlüsselten Nachricht bestätigen, wäre das auch bei ausbleibender Veröffentlichung der entschlüsselten Nachricht kein Beweis gegen das Bulletin Board– der Absender könnte den Schlüssel für sich behalten haben.

Natürlich sind hier noch viele weitere Angriffsszenarien denkbar, insbesondere auf die Verfügbarkeit: Das Bulletin Board kann ungültige Signaturen erzeugen, seine Dienste ganz einstellen usw. Da gegen diese Angriffe außer der einleitend erwähnten Verteilung auf mehrere Peers nichts getan werden kann, sind sie für den Entwurf eines Bulletin Boards aber nicht weiter interessant.

[15] Die Möglichkeit, dass das Bulletin Board die veröffentlichte Reihenfolge bzw. die veröffentlichten Zeitstempel nachträglich ändert, ist hier bewusst ausgeklammert: Ein solcher Angriff würde bereits die Append-Only-Eigenschaft verletzen.

4.3 Andere Arbeiten zu Bulletin Boards

In diesem Abschnitt werden verschiedene Ansätze bzw. Vorschläge zur Implementierung eines Bulletin Boards diskutiert. Soweit nicht explizit anders angegeben, benötigen alle vorgestellten Bulletin Boards, bei denen Signaturen verwendet werden, eine etablierte Public-Key-Infrastruktur, damit die Signaturen überprüft werden können. Zudem wird bei der Beschreibung von Angreifermodellen immer vorausgesetzt, dass der Angreifer keine kryptographischen Primitive wie z. B. Signaturen brechen kann.

4.3.1 Bulletin Board in Remotegrity/Gemeindewahl Takoma Park

In Zusammenarbeit mit der Wahlkommission von Takoma Park, Maryland, haben Zagórski et al. [109] ein eVoting-System namens *Remotegrity* entwickelt, im Rahmen einer Gemeindewahl (mit einigen Anpassungen) implementiert und mit freiwilligen Teilnehmern getestet – die eigentliche Wahl war unabhängig vom Test, wurde aber als Rahmen genutzt. Remotegrity ist – wie der Name schon andeutet – ein System, welches eine Art Briefwahl („remote“) erlaubt[16].

In der Beschreibung von Remotegrity wird einfach angenommen, dass es einen öffentlichen Broadcastkanal mit Append-Only-Eigenschaft gibt[17]. Für den erwähnten Testlauf wurde dann ein konkretes Bulletin Board benötigt. Dieses wurde wie folgt implementiert: Die Webserver, bei denen die Wähler ihre Stimme abgeben, schreiben direkt in eine (zentrale) Datenbank. Ein weiterer Server („Signing Daemon“) überwacht diese Datenbank auf neue Einträge hin und sobald ein neuer Eintrag auftaucht, wird er vom Signing Daemon gelesen, signiert und die Signatur in eine andere Tabelle (derselben Datenbank) geschrieben. Zusätzlich gibt es einige Beobachter, welche anhand der Signaturen überprüfen können, ob die Boardinhalte vollständig und unverändert sind.

Die wichtigen Eigenschaften Append-Only und Consistent View kann dieses Bulletin Board nur dann erfüllen, wenn sowohl der Signing Daemon als auch der Datenbankserver ehrlich sind; Receipts gibt es nicht einmal. Eine Verletzung kann nur bei Append-Only nachgewiesen werden und auch das nur, wenn der Signing Daemon als ehrlich angenommen wird – andernfalls könnte er ja einfach Nachrichten und zugehörige Signaturen erfinden.

Zagórski et al. merken dann auch in ihrer Zusammenfassung an, dass die Implementierung des Bulletin Boards ein Problem war und einige Sicherheitsanforderungen an das Bulletin Board aufgeweicht werden mussten (siehe Zitat auf Seite 49).

[16] In Takoma Park wurde in den normalen Wahlräumen bereits ein End-to-End Verifiable eVoting-System namens *Scantegrity II* eingesetzt [20, 21], welches physische Anwesenheit des Wählers erfordert.

[17] „Finally we assume the existence of an append-only broadcast channel, called a bulletin board.“ [109, S. 3]

4.3.2 Bulletin Board in Helios

Auch das *Helios* eVoting-System von Adida [2] wurde bereits (in verschiedenen Varianten) eingesetzt (z. B. von der IACR [50] und der Universität Louvain [3]) – und auch Helios benötigt ein Bulletin Board. In der ursprünglichen Variante wird dafür ein einfacher, zentraler Server verwendet – Adida begründet das damit, dass Helios mit dem Ziel entwickelt wurde, überprüfbar zu sein. Entsprechend schlägt Adida vor, dass auch das Bulletin Board dadurch „sicher“ wird, dass es von verschiedenen Parteien laufend auf Inkonsistenzen hin überprüft wird. Zudem sollen „genügend“ Wähler prüfen, ob ihre Stimmabgabe auf dem Bulletin Board veröffentlicht wird.

Problematisch mit Blick auf die gewünschten Eigenschaften eines Bulletin Boards ist bei Helios der Umstand, dass ein Angriff/Fehlverhalten des Bulletin Boards nicht *nachgewiesen* werden kann. Wenn z. B. eine der überprüfenden Parteien behauptet, ein Eintrag sei auf dem Bulletin Board veröffentlicht und kurz darauf wieder entfernt worden, gibt es für Dritte keine Möglichkeit, diese Behauptung zu verifizieren.

Allerdings merkt Adida selbst an, dass diese Variante, ein Bulletin Board zu implementieren, das „einfachste mögliche Bulletin Board“ ergibt und verweist – ohne konkrete Beispiele oder Arbeiten zu nennen – auf verteilte Bulletin Boards.

4.3.3 Einschub: Notation

Um den Nachrichtenfluss der im Folgenden diskutierten Bulletin Boards darstellen zu können, wird zusätzlich zu der aus Kapitel 2 bekannten die folgende Notation eingeführt:

Die Akteure teilen sich auf in Absender $w \in W$ von Nachrichten, Leser $r \in R$ und Peers $p \in P$ mit $|P| = N$. Falls eine Bezeichnung für einen bestimmten Akteur notwendig ist, werden die Akteure mit Indizes versehen, z. B. p_i. Dabei gilt außerdem $p_{j \neq i} := \{p_j \in P | j \neq i\}$. Zudem muss in einigen Fällen bei den Peers weiter unterschieden werden zwischen *Item Collectors* ($p \in P_{IC}$, $|P_{IC}| = N_{IC}$) und *Web Bulletin Board-Peers* ($wbb \in WBB$, $|WBB| = N_{WBB}$).

Weiter werden Nachrichten, die auf dem Bulletin Board veröffentlicht werden sollen, mit m bezeichnet. H^{λ}_{BB} ist der Hashwert über alle aktuell veröffentlichten Nachrichten des Bulletin Boards, analog dazu bezeichnet H^{i}_{BB} den Hashwert über die Nachrichten $m_1, ..., m_i$ in der Form, wie sie auf dem Bulletin Board veröffentlicht wurden – falls es eine totale Ordnung der Nachrichten gibt, wird sie dabei berücksichtigt.

Sendet eine Partei A die Nachricht m an Partei B, so wird dies als $A \rightarrow B : m$ notiert. Ansonsten gibt es nach wie vor private Schlüssel sk_Y, öffentliche Schlüssel pk_Y sowie Signaturen $\text{Sig}_Y(m)$ bzw. Thresholdsignaturen $\text{TSig}(x) = \text{combine}(\text{SigS}_{Y_1}(x), ..., \text{SigS}_{Y_k}(x))$.

4.3.4 Bulletin Board von Peters

Im Gegensatz zu den Arbeiten von Adida und Zagórski et al., die jeweils ein eVoting-System entwickelt haben, hat sich Peters [91] explizit mit der Implementierung eines Bulletin Boards beschäftigt. Er untersucht mehrere Multicast-Protokolle daraufhin, wie gut sie als zentrale Einheit eines verteilten Bulletin Boards verwendet werden können; der Fokus liegt dabei auf verschiedenen Performance-Eigenschaften wie Kommunikations- und Zeitkomplexität. Basierend auf diesem Vergleich wählt Peters dann eine leicht angepasste Variante des Rampart-Protokolls [92] aus und implementiert damit ein entsprechendes Bulletin Board.

Die Kommunikation zum Veröffentlichen einer Nachricht läuft dabei wie folgt ab ($k > \frac{2}{3}N$)[18]:

$$w \longrightarrow p_i \quad : m \tag{PP1}$$

$$p_i \longrightarrow p_{j\neq i} : h(m) \tag{PP2}$$

$$p_{j\neq i} \longrightarrow p_i \quad : l, \text{SigS}_{p_j}(p_i, h(m), l) \tag{PP3}$$

$$p_i \quad : \text{TSig}(p_i, h(m), l) = \text{combine}\left(\left\{\text{SigS}_{p_j}(p_i, h(m), l)\right\}_{p_j \in P' \subseteq P, |P'| \geq k}\right) \tag{PP4}$$

$$p_i \longrightarrow p_{j\neq i} : m, \text{TSig}(p_i, h(m), l) \tag{PP5}$$

$$p_{j\neq i} \longrightarrow p_i \quad : \text{SigS}_{p_j}(m) \tag{PP6}$$

$$p_i \quad : \text{TSig}(m) = \text{combine}\left(\left\{\text{SigS}_{p_j}(m)\right\}_{p_j \in P'' \subseteq P, |P''| \geq k}\right) \tag{PP7}$$

$$p_i \longrightarrow w \quad : \text{TSig}(m) \tag{PP8}$$

Der Sinn der Schritte (PP2) und (PP3) ist, dass sich die Peers $p_{j\neq i}$ dazu verpflichten, dass sie die Nachricht m als $l$te Nachricht annehmen werden. In (PP5) erhalten die Peers $p_{j\neq i}$ dann eine Bestätigung, dass genügend andere Peers sich verpflichtet haben, m anzunehmen – damit wird sichergestellt, dass entweder mindestens k oder keiner der Peers m annehmen. Anschließend erhält p_i Teilsignaturen über m von den anderen Peers und kann aus diesen dann ein Receipt für den Absender erzeugen.

Beim Lesen der Boardinhalte stellt sich der Nachrichtenfluss folgendermaßen dar:

$$r \longrightarrow p_i \quad : \text{Nonce } n_r \tag{PR1}$$

$$p_i \longrightarrow p_{j\neq i} : n_r \tag{PR2}$$

$$p_{j\neq i} \longrightarrow p_i \quad : \text{SigS}_{p_j}(n_r, m_1, ..., m_l) \tag{PR3}$$

$$p_i \quad : \text{TSig}(n_r, m_1, ..., m_l) = \text{combine}\left(\left\{\text{SigS}_{p_j}(n_r, m_1, ...)\right\}_{p_j \in P' \subseteq P, |P'| \geq k}\right) \tag{PR4}$$

$$p_i \longrightarrow r \quad : m_1, ..., m_l, \text{TSig}(n_r, m_1, ..., m_l) \tag{PR5}$$

[18] Dabei wurden einige Details des zugrunde liegenden Multicast-Protokolls der Verständlichkeit halber ausgelassen; das gilt auch für den Nachrichtenfluss beim Lesen.

Die Peers erstellen also gemeinsam eine Thresholdsignatur über den aktuellen Boardinhalt. Zusammen mit den Receipts sind also Verletzungen von Append-Only, Receipts und Consistent View nachweisbar. So lange außerdem mehr als zwei Drittel der Peers ehrlich sind, können neben diesen auch Zensurfreiheit und Verfügbarkeit garantiert werden. Allerdings muss das Angreifermodell neben der Schranke für kompromittierte Peers noch weiter eingeschränkt werden: Weil das Rampart-Protokoll auf einem sogenannten *Secure Membership Protocol* aufbaut, welches „fehlerhafte" Peers aus dem Protokoll ausschließt – und dabei gilt schon eine verspätete Antwort als Fehler – darf der Angreifer Nachrichten höchstens eine sehr kurze Zeit verzögern. Ansonsten könnte der Angreifer alle ehrlichen Peers aus dem Protokoll „werfen".

4.3.5 Bulletin Board von Hauser und Haenni

Hauser und Haenni beschreiben in [53] unter anderem eine Reihe von Anforderungen und Herausforderungen für den Entwurf eines Bulletin Boards. Während die formulierten Herausforderungen wie z. B. „Archivieren einer abgeschlossenen Wahl" recht konkret sind, ist ihr Vorschlag für den Entwurf eines Bulletin Boards eher abstrakt (z. B. gehen sie nicht darauf ein, ob und wie ein entsprechendes Bulletin Board verteilt implementiert werden könnte).

Um eine Nachricht zu veröffentlichen, ist folgende Kommunikation notwendig[19], dabei hat das Bulletin Board bisher $i-1$ Nachrichten $m_1, ..., m_{i-1}$ veröffentlicht:

$$w \longrightarrow p \ : pk_w, m, \text{Sig}_w(m) \tag{HP1}$$
$$p \longrightarrow w : i, T_{BB}, h(m_{i-1}), \text{Sig}_{BB}\,(m, i, T_{BB}, h(m_{i-1})) \tag{HP2}$$

Die Kommunikation zum Lesen von Boardinhalten sieht folgendermaßen aus, dabei ist Q eine Art Filter, damit nicht immer der komplette Boardinhalt auf einmal gelesen werden muss, $\mathcal{R}$ ist die Menge der zum Filter passenden Nachrichten auf dem Bulletin Board:

$$r \longrightarrow p : Q \tag{HR1}$$
$$p \longrightarrow r : T_{BB}, \mathcal{R}, \text{Sig}_{BB}\,(Q, \mathcal{R}, T_{BB}) \tag{HR2}$$

Durch den Hashwert $h(m_{i-1})$ in (HP2) entsteht eine Hashkette der Einträge auf dem Bulletin Board und so implizit eine totale Ordnung; durch die (signierten) Zeitstempel in (HP2) und (HR2) können Leser und Absender durch Vergleich der Zeitstempel außerdem nachweisen, falls das Bulletin Board die Veröffentlichung einer Nachricht trotz ausgestelltem Receipt verzögert[20].

[19] Auch hier werden wieder einzelne Details ausgelassen: Hauser und Haenni sehen vor, dass das Bulletin Board in Gruppen und Sektionen unterteilt ist, dementsprechend enthalten die Nachrichten bei ihnen noch Informationen darüber, zu welcher Gruppe und Sektion die jeweilige Nachricht gehört.

[20] Unter der Annahme, dass das Bulletin Board korrekte Zeitstempel ausstellt – sollte das Bulletin Board falsche Zeitstempel ausstellen, könnte das beim Schreiben nachweisbar sein, falls die Ordnung der Zeitstempel nicht mit der durch die Hashkette induzierten Reihenfolge übereinstimmt. Ansonsten können – ein teilsynchrones Netzwerkmodell vorausgesetzt – Zeitstempel „aus der Zukunft" als Beweis für einen Betrug des Bulletin Boards genutzt werden, wenn sie schnell genug (d. h. so lange der Zeitstempel noch „in der Zukunft" liegt) geprüft werden.

Damit können diesem Bulletin Board prinzipiell Verletzungen der Eigenschaften Append-Only, Receipts, Consistent View und totale Ordnung nachgewiesen werden. Allerdings beschreiben Hauser und Haenni nicht, wie ein solches Bulletin Board implementiert werden könnte und sie definieren kein Angreifermodell.

4.3.6 Bulletin Board in D-DEMOS

Für ihr eVoting-System D-DEMOS [25] haben Chondros et al. auch ein Bulletin Board entwickelt, das sie auf zwei Subsysteme verteilen: Die sogenannten *Vote Collectors* und das *BB Subsystem*, beides sind jeweils verteilte Systeme. Die Peers im BB Subsystem sind dabei völlig unabhängig voneinander und dienen ausschließlich zum Speichern der Boardinhalte, die ihnen von den Vote Collectors geliefert werden. Die Vote Collectors wiederum sind dafür zuständig, die Stimmabgaben der Wähler zu sammeln, Receipts dafür auszustellen und sich am Ende der Wahlperiode auf eine einheitliche Menge an abgegebenen Stimmen zu einigen.

Dass sich die Vote Collectors erst am Ende der Wahlperiode auf eine Menge an abgegebenen Stimmen einigen, liegt daran, dass diese „Einigung“ ein recht aufwändiges Protokoll erfordert – dieses Protokoll für jede einzelne Stimme (oder in regelmäßigen Abständen) auszuführen, wäre zu teuer bezüglich der Performance des Systems. Damit trotzdem sofort Receipts ausgestellt werden können, erhält jeder Vote Collector p_i bereits vor Beginn der Wahl einen „Receipt-Anteil“ rs_i für jede mögliche Stimmabgabe[21]. Die Stimmabgabe läuft dann wie folgt ab, dabei ist $k > \frac{2}{3}N_{IC}$:

$$w \longrightarrow p_i \quad : m \tag{DV1}$$
$$p_i \quad : \text{validiert } m \tag{DV2}$$
$$p_i \longrightarrow p_{j\neq i} \quad : m \tag{DV3}$$
$$p_{j\neq i} \longrightarrow p_i \quad : m, \text{Sig}_{p_j}(m) \tag{DV4}$$
$$p_i \quad : \mathsf{UCERT}(m) = \left\{\text{Sig}_{p_j}(m)\right\}_{p_j \in P' \subseteq P, |P'| \geq k} \tag{DV5}$$
$$p_i \longrightarrow p_{j\neq i} \quad : m, \mathsf{UCERT}(m), rs_i \tag{DV6}$$
$$p_{j\neq i} \longrightarrow p_{k\neq j} \quad : m, \mathsf{UCERT}(m), rs_j \tag{DV7}$$
$$p \in P \quad : \text{konstruiert Receipt } \mathcal{R} \text{ aus } \{rs_j\}_{p_j \in P' \subseteq P, |P'| \geq k} \tag{DV8}$$
$$p_i \longrightarrow w \quad : \mathcal{R} \tag{DV9}$$

Durch die Verwendung von $\mathsf{UCERT}(m)$ wird sichergestellt, dass genügend Peers bereit sind, die Nachricht m anzunehmen, bevor sie ihren Receipt-Anteil preisgeben – ähnlich wie bei Peters.

Nach Ende der Wahlperiode führen die Vote Collectors dann ein weiteres Protokoll aus, um sich auf einen Boardinhalt zu einigen. Weil dieses Protokoll eng mit dem konkreten Inhalt

[21] Chondros et al. verwenden zum Aufteilen der Receipts ein $(N - f, N)$ *Verfiable Secret-Sharing*-Schema, dabei wird ein Geheimnis über N Parteien verteilt, zur Wiederherstellung braucht es dann die Anteile von mindestens $N - f$ dieser Parteien.

der Nachrichten und damit dem D-DEMOS-System zusammenhängt, wird es hier nicht weiter diskutiert. Nachdem sich die Vote Collectors auf einen Boardinhalt geeinigt haben, sendet jeder Vote Collector den Boardinhalt an alle Peers im BB Subsystem[22]. Sobald ein Peer des BB Subsystems von mindestens zwei Dritteln der Vote Collectors denselben Boardinhalt erhalten hat, veröffentlicht er diesen[23].

Der Angreifer darf bei Chondros et al. Nachrichten bis zu einer gewissen Grenze verzögern, weniger als ein Drittel der Vote Collectors und weniger als die Hälfte der Peers im BB Subsystem kompromittieren. Außerdem nehmen die Autoren an, dass es geheime, authentifizierte Kanäle zwischen den Vote Collectors sowie authentifizierte Kanäle zwischen Vote Collectors und Peers im BB Subsystem gibt. Unter diesen Annahmen erfüllt das Bulletin Board die folgenden Eigenschaften bzw. kann ihm die Verletzung dieser nachgewiesen werden: Append-Only, Receipts, Consistent View[24], Verfügbarkeit und Zensurfreiheit. Eine totale Ordnung der Nachrichten wird nicht gefordert.

4.3.7 Bulletin Board von Culnane und Schneider

Für das von Culnane et al. entwickelte eVoting-System vVote [29] haben Culnane und Schneider in [30] ein verteiltes Bulletin Board vorgestellt und seine Eigenschaften in einem Event-B-Modell untersucht. Eine auffällige Besonderheit an diesem Bulletin Board ist, dass der Boardinhalt weder kontinuierlich noch am Ende der Wahlperiode, sondern in regelmäßigen Abständen veröffentlicht wird[25]. Im Folgenden werden die Zeiträume von einer Veröffentlichung des Boardinhalts bis zur nächsten als Zeitabschnitte bezeichnet. Wie beim Bulletin Board von Chondros et al. ist auch hier Performance der Grund dafür, dass nicht kontinuierlich veröffentlicht wird.

Um eine Nachricht innerhalb von Zeitabschnitt a zu veröffentlichen, nimmt der Absender an folgendem Protokoll teil ($k > \frac{2}{3}N$):

$$w \longrightarrow p \in P : m \tag{CSP1}$$

$$p_i \in P \longrightarrow p_{j \neq i} : a, m, \text{Sig}_{p_i}(a, m) \tag{CSP2}$$

$$p_i \in P : \text{wartet auf } \geq k \text{ Nachrichten } \text{Sig}_{p_j}(a, m) \text{ und speichert sie in } D_{i,a} = \{\text{Sig}_{p_{j_1}}(a, m), \text{Sig}_{p_{j_2}}(a, m), ..., \text{Sig}_{p_{j'_1}}(a, m'), ...\} \tag{CSP3}$$

$$p_i \in P \longrightarrow w : \text{SigS}_{p_i}(a, m) \tag{CSP4}$$

$$w : \text{TSig}(a, m) = \text{combine}\left(\left\{\text{SigS}_{p_j}(a, m)\right\}_{p_j \in P' \subseteq P, |P'| \geq k}\right) \tag{CSP5}$$

[22] Chondros et al. fordern nicht, dass die Vote Collectors den Boardinhalt signieren, sie nehmen aber an, dass es authentifizierte Kanäle zwischen den Vote Collectors und den Peers des BB Subsystems gibt

[23] Die Peers des BB Subsystems haben bei Chondros et al. noch mehr Aufgaben – diese zusätzlichen Aufgaben betreffen aber das eVoting-System und nicht die hier betrachtete Bulletin Board-Funktionalität.

[24] Unter der Annahme, dass beim Lesen von allen Peers des BB Subsystems gelesen und die Antwort der Mehrheit übernommen wird – die Autoren beschreiben den Lesevorgang nicht.

[25] Konkret schlagen die Autoren vor, einmal täglich zu veröffentlichen – im Kontext der *Victoian State Election*, bei der vVote zum Einsatz kam [18], ergibt das Sinn, weil es dort eine zweiwöchige „Vorwahlphase“ gibt.

Auch hier warten die Peers in (CSP3) darauf, dass genügend andere Peers ankündigen, die Nachricht anzunehmen, bevor sie in (CSP4) ihren Teil des Receipts herausgeben.

Am Ende jedes Zeitabschnitts müssen sich die Peers dann auf einen Boardinhalt einigen, dafür stellen die Autoren zwei Protokolle vor: Zunächst wird ein effizientes optimistisches Protokoll ausgeführt – *optimistisch*, weil es voraussetzt, dass mindestens t ehrliche Peers die ganze Zeit online waren. Schlägt das optimistische Protokoll fehl, wird ein *Fallback*-Protokoll ausgeführt, um die lokalen Boards zu synchronisieren; danach wird nochmals das optimistische Protokoll verwendet, um die nötigen Signaturen zu erzeugen. Das optimistische Protokoll funktioniert wie folgt, dabei ist $B_{i,a}$ der Boardinhalt von p_i für Zeitabschnitt a:

$$p_i \in P \longrightarrow p_{j\neq i} : \text{Sig}_{p_i}(a, B_{i,a}) \tag{CSO1}$$

$$p \in P : \text{Wartet auf } \geq k \text{ Signaturen } \text{Sig}_{p_j}(a, B_{j,a}) \text{ mit identischen } B_{j,a} \tag{CSO2}$$

$$p_i \in P \longrightarrow wbb : B_{i,a}, \text{SigS}_{p_i}(a, B_{i,a}) \tag{CSO3}$$

Im letzten Schritt schicken alle Peers ihren Boardinhalt zusammen mit einer Teilsignatur an ein „Web Bulletin Board“ wbb, dieses erstellt dann aus den Teilsignaturen eine vollständige Signatur und veröffentlicht den Boardinhalt. Culnane und Schneider beschreiben das wbb nicht genauer, es wäre aber sinnvoll, eine ähnliche Konstruktion zu verwenden, wie sie von Chondros et al. für deren BB Subsystem vorgeschlagen wurde.

Erhält ein Peer in Schritt (CSO2) nicht genügend Nachrichten mit identischen $B_{j,a}$, dann wird das folgende Fallback-Protokoll verwendet, um die lokalen Boardinhalte zu synchronisieren:

$$p_i \in P \longrightarrow p_{j\neq i} : D_{i,a} \text{ aus Schritt (CSP3)} \tag{CSF1}$$

$$p_i \in P : D_{i,a} = \bigcup_{p_j \in P} D_{j,a} \tag{CSF2}$$

$$p_i \in P : \forall m_l : \left|\left\{\text{Sig}_{p_j}(m_l, a) \in D_{i,a}\right\}\right| \geq k \Rightarrow m_l \in B_{i,a} \tag{CSF3}$$

Nachdem die Peers in (CSF3) ihren lokalen Boardinhalt synchronisiert haben, wird erneut das optimistische Protokoll ausgeführt.

Als Angreifer erlauben Culnane und Schneider einen Dolev-Yao-Angreifer [32], d. h. der Angreifer hat volle Kontrolle über das Netzwerk. Zudem darf der Angreifer beliebige Peers kompromittieren, so lange es weniger als ein Drittel aller Peers sind – allerdings gibt es Einschränkungen bezüglich des Fallback-Protokolls, werden diese nicht eingehalten, können sich die Peers ggf. nicht auf einen Boardinhalt einigen (und wiederholen die Protokolle endlos).

Unter den genannten Voraussetzungen erfüllt das vorgeschlagene Bulletin Board Append-Only, Receipts, Consistent View[26], und Zensurfreiheit. Eine totale Ordnung ist nicht vorgesehen. Wegen Verteilung und der Möglichkeit, Nachrichten parallel zu verarbeiten, darf für dieses Bulletin Board eine vergleichsweise gute Performance und Verfügbarkeit erwartet werden (solange die Kommunikation nicht vom Angreifer unterbunden wird).

[26] Culnane und Schneider beschreiben nicht, wie die veröffentlichten Daten gelesen werden können – sinnvoll wäre, den gesamten Boardinhalt plus die Signatur zu lesen und dann die Signatur zu prüfen. Davon wird hier ausgegangen.

4.3.8 Bulletin Board von Kuldmaa

Ausgehend von einem etwas stärkeren Angreifermodell untersucht Kuldmaa [70] das Bulletin Board von Culnane und Schneider, identifiziert einige Schwachstellen und schlägt eine Variante des Bulletin Boards von Culnane und Schneider vor, die auch im von Kuldmaa formulierten Angreifermodell sicher ist.

Am Bulletin Board von Culnane und Schneider nimmt sie dafür im Wesentlichen zwei Änderungen vor: Die Erste betrifft Schritt (CSP2): Bei Culnane und Schneider ist unklar, wie sich ein Peer verhalten soll, wenn er $a, m, \mathrm{Sig}_{p_i}(a, m)$ erhält, ohne vorher die Nachricht aus (CSP1) erhalten zu haben. Kuldmaa schlägt daher vor, dass ein Peer in dieser Situation m aus der (CSP2)-Nachricht entnimmt und damit seinerseits in Schritt (CSP2) einsteigt. Damit wird sichergestellt, dass auch Nachrichten, deren Absender sie nicht an alle Peers versendet haben, bei allen ehrlichen Peers ankommen.

Die zweite Änderung ist, dass die Peers vor der Veröffentlichung keine signierten Boardinhalte mehr austauschen, sondern ihren Boardinhalt direkt mit einer Teilsignatur an das Web Bulletin Board senden – d. h. das optimistische und das Fallback-Protokoll fallen weg. Der Grund dafür ist, dass wegen der ersten Änderung ohnehin alle ehrlichen Peers denselben Boardinhalt haben sollten – dementsprechend ist es nicht notwendig, diesen nochmals zu „synchronisieren“. Das Veröffentlichen funktioniert daher bei Kuldmaa anders:

$$p_i \in P \longrightarrow wbb : i, a, B_{i,a}, \mathrm{SigS}_{p_i}(a, B_{i,a}) \tag{KP1}$$

$$wbb : \mathrm{TSig}(a, B_a) = \mathrm{combine}\left(\left\{\mathrm{SigS}_{p_j}(a, B_a)\right\}_{p_j \in P' \subseteq P, |P'| \geq k}\right) \tag{KP2}$$

Diese Änderungen erlauben es Kuldmaa, die Einschränkungen für den Angreifer während des Fallback-Protokolls fallen zu lassen. Bezüglich der sonstigen Eigenschaften des Bulletin Boards gilt dasselbe wie bei Culnane und Schneider, lediglich das Angreifermodell ist hier etwas stärker.

4.3.9 Bulletin Board von Heather und Lundin

Ein weiteres Bulletin Board wird von Heather und Lundin in [55] vorgeschlagen. Eine Besonderheit dieses Bulletin Boards ist, dass die Reihenfolge der Nachrichten durch die Absender der Nachrichten festgelegt wird. Zudem wurde beim Entwurf bereits berücksichtigt, dass Uhren in einem verteilten System nicht komplett synchron laufen und dass Nachrichten eine gewisse Zeit „unterwegs“ sind, bevor sie ankommen. Darum gibt es in den Protokollbeschreibungen hier eine Schranke ε für die Differenz zwischen zwei Zeitstempeln – wird diese überschritten, wird das Protokoll an dieser Stelle abgebrochen; notiert wird eine solche Prüfung als „$|T_1 - T_2| \overset{?}{<} \varepsilon$“. Die Zeitstempel werden jeweils in dem Moment „erstellt“, in dem sie das erste Mal im Protokoll auftauchen.

Um eine Nachricht m zu veröffentlichen, ist folgendes Protokoll vorgesehen (da die Autoren zunächst von einem zentralen Bulletin Board ausgehen, wird auch hier vorerst von nur einem Peer p ausgegangen):

$$w \longrightarrow p \ : \text{Lese } H_{BB}^{\lambda} \tag{HLP1}$$

$$p \longrightarrow w : H_{BB}^{\lambda}, T_{BB}, \text{Sig}_{BB}\left(H_{BB}^{\lambda}, T_{BB}\right) \tag{HLP2}$$

$$w \qquad : |T_w - T_{BB}| \overset{?}{<} \varepsilon \tag{HLP3}$$

$$w \qquad : H_w := h\left(m, T_w, w, H_{BB}^{\lambda}\right) \tag{HLP4}$$

$$w \longrightarrow p \ : m, T_w, w, H_w, \text{Sig}_w(H_w) \tag{HLP5}$$

$$p \qquad : |T_w - T_{BB}| \overset{?}{<} \varepsilon \tag{HLP6}$$

$$p \longrightarrow w : T'_{BB}, \text{Sig}_{BB}\left(\text{Sig}_w(H_w), T'_{BB}\right) \tag{HLP7}$$

$$w \qquad : |T'_{BB} - T_w| \overset{?}{<} \varepsilon \tag{HLP8}$$

Dadurch, dass der Hashwert H_{BB}^{λ} Teil seiner Nachricht ist, legt der Absender einer Nachricht fest, nach welchen anderen Nachrichten seine eigene veröffentlicht werden soll, d. h., die Ordnung der Nachrichten wird durch die Absender definiert. Die Prüfung in (HLP8) erscheint auf den ersten Blick unnötig, tatsächlich kann w auch an dieser Stelle die Veröffentlichung noch abbrechen: Ist $|T'_{BB} - T_w| < \varepsilon$ erfüllt, dann hat w mit der Nachricht aus Schritt (HLP7) ein Receipt und könnte dem Bulletin Board Betrug nachweisen, falls m nicht veröffentlicht wird. Ist dagegen $|T'_{BB} - T_w| < \varepsilon$ *nicht* erfüllt, dann dient die gleiche Nachricht als Nachweis dafür, dass m nicht hätte veröffentlicht werden dürfen, weil das Bulletin Board durch seine Signatur sowohl T'_{BB} als auch T_w unterschrieben hat.

Gelesen werden kann der Boardinhalt $m_1, ..., m_l$ wie folgt:

$$r \longrightarrow p : \text{Lese Boardinhalt} \tag{HLR1}$$

$$p \longrightarrow r : m_1, ..., m_l, \text{Sig}_{BB}\left(H_{BB}^{\lambda}, T_{BB}\right) \tag{HLR2}$$

Bezüglich der gewünschten Eigenschaften erfüllt dieses Bulletin Board (ggf. im Sinne von Nachweisbarkeit bei Verletzung): Append-Only, Receipts, Consistent View und totale Ordnung. Zensurfreiheit und Verfügbarkeit sind nur dann garantiert, wenn das Bulletin Board ehrlich ist.

Wie oben angemerkt stellen die Autoren das Bulletin Board zunächst als zentralisiertes System vor. Um das System robuster bzw. „schneller“ zu machen, schlagen sie anschließend noch zwei verteilte Varianten vor.

Bei der ersten Variante, „synchronized“ genannt, gibt es N Peers, die jeweils einen Schlüsselteil für eine k-aus-N Thresholdsignatur mit $k > \frac{N}{2}$ haben. Ein Absender kommuniziert dann mit einem dieser Peers wie oben beschrieben, dieser Peer versucht jetzt aber über ein Locking-Protokoll, die nötigen Teilsignaturen von weiteren $k - 1$ Peers zu erhalten – dabei fügen auch diese anderen Peers die Nachricht in ihren Boardinhalt ein (und übermitteln sie anschließend

an die restlichen Peers). Ein großer Nachteil dieser Variante ist, dass alle Peers zusammen zu einem Zeitpunkt nur eine Nachricht verarbeiten können – andernfalls könnte es passieren, dass Signaturen und damit Receipts für mehrere Nachrichten erzeugt werden, die alle denselben Hashwert H_{BB}^{λ} enthalten (und folglich an derselben Stelle in der Nachrichtenfolge eingefügt werden müssten). Dafür können hier einige Peers ausfallen, ohne dass das System an sich ausfällt – die Autoren untersuchen jedoch nicht weiter, wie viele genau und ob auch Byzantinische Fehler toleriert werden.

Um die Performance zu steigern, schlagen die Autoren noch eine zweite Variante, „unsynchronized“, vor. Dabei wird die Forderung nach einer totalen Ordnung über alle Peers hinweg fallen gelassen, die einzelnen Peers haben allerdings weiterhin eine solche. Aus Sicht eines Absenders ändert sich auch hier nichts: Der Peer, an den er seine Nachricht sendet, versucht $k - 1$ weitere Peers für eine gemeinsame Signatur zu „finden“ und kann damit dann ein Receipt ausstellen. Weil eine einzelne Nachricht nun nicht mehr zwingend auf allen, sondern nur noch auf k Peers veröffentlicht wird, müssen beim Lesen entsprechend mindestens $N - k + 1$ Peers gelesen werden, um alle Nachrichten zu sehen.

Bei Heather und Lundin werden für die „unsynchronized“-Variante keine weiteren Änderungen gegenüber ihrem initialen Vorschlag genannt, Krummenacher [68] merkt aber (zurecht) an, dass noch mehr geändert werden müsste: Weil der Absender sich auf einen Wert H_{BB}^{λ} in seiner Nachricht festlegen muss, müsste es mindestens k Peers mit identischem Boardinhalt geben, andernfalls kann die Nachricht nicht auf k Peers – und somit gar nicht – veröffentlicht werden. So wie Heather und Lundin die Variante vorstellen, ist das aber weder sichergestellt noch sonderlich wahrscheinlich (ihre Motivation für diese Variante ist ja gerade, dass mehrere Absender zugleich Nachrichten veröffentlichen können).

5 Implementierungsmodelle für Bulletin Boards auf Basis von Blockchains

Nachdem in den letzten beiden Kapiteln Blockchains und Bulletin Boards mit ihren jeweiligen Eigenschaften diskutiert und einige „klassische“ Bulletin Boards vorgestellt wurden, wird in diesem Kapitel die Frage behandelt, wie die beiden zusammengebracht werden können. Die Motivation dafür ist die auf den ersten Blick enge Verwandtschaft bei grundlegenden Eigenschaften: Sowohl bei Blockchains als auch bei Bulletin Boards ist es erklärtes Ziel, dass einmal „enthaltene“ Daten – also Transaktionen und Blöcke bei einer Blockchain bzw. veröffentlichte Nachrichten bei einem Bulletin Board– nicht mehr entfernt oder verändert werden können. Weiter soll in beiden Fällen erreicht werden, dass alle im weitesten Sinne Beteiligten dieselben Daten „sehen“.

Weil ein Bulletin Board per Definition öffentlich sein soll, werden hier nur öffentliche Blockchains betrachtet – das hat wie in Kapitel 3 dargelegt noch einen weiteren wichtigen Vorteil: Nichtöffentliche – also permissioned – Blockchains sind in der Regel nicht komplett dezentral, weil es eine Instanz geben muss, die bestimmt, wer teilnehmen darf und wer nicht. Wie in Kapitel 4 dargelegt, ist Abhängigkeit von einer zentralen Instanz bei einem Bulletin Board nachteilig für Sicherheit und Verfügbarkeit. Und selbst wenn die Entscheidung, wer teilnehmen darf und wer nicht, dezentral getroffen wird, widerspricht das dem Grundgedanken eines öffentlichen Boards, auf dem jeder Nachrichten veröffentlichen kann.

Weiter wird in diesem Kapitel gelegentlich die Bezeichnung „größere öffentliche Blockchain“ verwendet. Gemeint ist damit eine populäre Blockchain wie – zum aktuellen Zeitpunkt – Bitcoin oder Ethereum. Diese zeichnen sich unter anderem dadurch aus, dass zum einen der finanzielle Gegenwert der implementierten digitalen Währung sehr groß ist und zum anderen der Konsens-Algorithmus von entsprechend vielen Parteien ausgeführt und überprüft wird. Der Vorteil, der daraus für ein Bulletin Board resultiert, ist die höhere Sicherheit: Ein Angriff auf eine solche Blockchain ist deutlich teurer als er es bei einer „kleinen“ Blockchain wäre. Dementsprechend sollte ein Bulletin Board auch nicht auf Basis einer Blockchain gebaut werden, die es nur wegen dieses einen Bulletin Boards gibt. Die Teilnehmer an einer solchen Blockchain könnten dann nämlich auch einfach Peers eines verteilten Bulletin Boards sein, die möglichen Vorteile durch Verwendung einer öffentlichen Blockchain wären damit weitgehend hinfällig. Insbesondere sollte die Blockchain, auf der ein Bulletin Board aufgebaut wird, immer weiter wachsen. Soweit nicht anders erwähnt, wird für eine Blockchain in diesem Kapitel angenommen, dass sie die Eigenschaften Persistenz und Lebendigkeit wie in Abschnitt 3.8 beschrieben erfüllt.

Im Folgenden werden zunächst nochmals die Eigenschaften von klassischen Bulletin Boards rekapituliert. Daran schließt sich eine grundsätzliche Diskussion über die Nachweisbarkeit von Änderungen an einer Blockchain an. Anschließend werden einige Möglichkeiten diskutiert, ein Bulletin Board auf Basis einer Blockchain zu implementieren. Den Anfang macht dabei die einfachste Variante: Eine Blockchain wird direkt als Bulletin Board verwendet. Darauf folgt ein Überblick über hybride Ansätze aus einem klassischem Bulletin Board zusammen mit einer Blockchain und eine Bewertung dieser Ansätze. Abgeschlossen wird das Kapitel mit einer kurzen Einführung und Diskussion, inwiefern sich *Smart Contracts* zur Implementierung eines Bulletin Boards nutzen lassen.

5.1 Zentrales bzw. verteiltes System als Bulletin Board

Die in Abschnitt 4.3 beschriebenen Bulletin Board-Systeme verwenden entweder einen zentralen Server oder ein verteiltes System von Peers, um die nötige Funktionalität zu implementieren. Da deren Vor- und Nachteile bereits in Abschnitt 4.3 diskutiert wurden, wird hier nur kurz wiederholt, unter welchen Voraussetzungen und wie gut sie jeweils (typischerweise) die geforderten Eigenschaften erfüllen. In beiden Fällen wird für die Prüfung von Signaturen eine Public-Key-Infrastruktur vorausgesetzt.

Zentraler Server Damit bei einem zentralen Server Zensurfreiheit erfüllt werden kann, darf dieser Server nicht kompromittiert sein. Für sichere Verfügbarkeit muss der Server zudem erstens zuverlässig funktionieren und zweitens muss das Netzwerk mindestens teilsynchron sein. Falls alle relevanten Nachrichten des Bulletin Boards an andere Parteien signiert werden, sind Receipts möglich und die Verletzung der Eigenschaften Receipts, Consistent View und Append-Only kann jeweils zweifelsfrei nachgewiesen werden. Bezüglich einer totalen Ordnung hängt es vom genauen Schreibprotokoll ab: Wird die Ordnung vom Bulletin Board festgelegt, dann kann ein einzelner Server Nachrichten beliebig verzögern, ohne dass ihm das nachgewiesen werden kann. Wird dagegen wie bei Heather und Lundin [55] die Reihenfolge durch die Absender festgelegt, kann eine solche Verzögerung unter bestimmten Umständen nachgewiesen werden (für Details siehe Unterabschnitt 4.3.9). Die Performance des Bulletin Boards hängt im Falle eines einzelnen Servers natürlich an genau diesem. Auf Seiten der Absender und Leser von Nachrichten sind bei einem zentralen Bulletin Board in der Regel nur Prüfungen der Signaturen und bei Absendern ggf. auch die Erzeugung von Signaturen notwendig.

Verteiltes Bulletin Board Bei einer verteilten Implementierung muss das Netzwerk ebenfalls teilsynchron sein, um Verfügbarkeit gewährleisten zu können. Für Nachweise bei Verletzung der Eigenschaften Receipts, Consistent View und Append-Only genügt es auch hier, wenn das Bulletin Board alle relevanten Nachrichten signiert. Damit das Bulletin Board auch dann noch verfügbar ist, wenn einzelne Peers nicht mehr verfügbar sind, müssen zu jedem Zeitpunkt mindestens so viele Peers verfügbar sein, wie es braucht, um eine Signatur zu erstellen – üblicherweise mehr als zwei Drittel der Peers. Für die meisten verteilten Bulletin Boards gilt

weiter, dass Zensurfreiheit garantiert werden kann, so lange weniger als ein Drittel der Peers kompromittiert ist. Eine totale Ordnung garantieren die verteilten Systeme normalerweise nicht und wenn, dann braucht es auch dafür mehr als zwei Drittel ehrliche Peers[1]. Bezüglich der Performance sind verteilte Bulletin Boards in der Regel den zentralisierten Lösungen deutlich überlegen, was die Anzahl der pro Zeiteinheit verarbeiteten Nachrichten angeht. Die Leser und Absender von Nachrichten müssen Signaturen prüfen und diese ggf. vorher noch aus Teilsignaturen zusammensetzen; je nach System müssen die Absender auch hier ihre Nachrichten signieren.

5.2 Nachweisbarkeit von nachträglichen Änderungen bei Blockchains

Bevor ein genauerer Blick auf die einzelnen Implementierungsmodelle für ein Bulletin Board auf Basis einer Blockchain geworfen wird, muss eine wichtige Beobachtung festgehalten werden: Bei einer Blockchain gibt es keine Möglichkeit, eine nachträgliche Änderung unabstreitbar nachzuweisen. Insbesondere reicht es nicht, gültige Nachfolger-Blöcke (Confirmations) zu präsentieren: Wenn die Mehrheit der Miner gemeinsam einen Angriff ausführt, sind die Confirmations komplett wertlos (im Gegensatz zu Threshold-Signaturen). Das kann man sich für Proof of Work-Blockchains am Beispiel in Abbildung 5.1 intuitiv klar machen: Ein Teilnehmer A sieht den Block B_2 mit TX_1 als Teil des active Fork, diese Sicht wird anschließend noch durch zwei Confirmations bestätigt (B_3 und B_4). Doch dann tauchen – z. B. durch Verzögerungen im Netzwerk oder einen bewussten Angriff – die Blöcke B_5 bis B_8 auf. Die Teilnehmer (also auch A) übernehmen den neuen Fork B_5, B_6, B_7, B_8 als active Fork und „entfernen“ damit TX_1 aus der Blockchain. Wenn nun A diese nachträgliche Änderung gegenüber einem Dritten B beweisen möchte, könnte A die Blöcke B_2 bis B_4 als „Beweis“ vorlegen – das Problem ist, dass A diese Blöcke auch selbst hätte erzeugen können. Aus Sicht von B ist damit insbesondere nicht zweifelsfrei bewiesen, dass TX_1 jemals Teil des active Fork war.

Für ein Bulletin Board, das (nur) aus einer Blockchain besteht, folgt daraus, dass es keine Nachweise für nachträgliche Änderungen geben kann. Das bedeutet, dass entgegen der Forderungen aus Abschnitt 4.1 keine Verletzungen von Append-Only, Consistent View und totaler Ordnung nachweisbar sind. In den folgenden Abschnitten liegt der Fokus darum darauf, unter welchen Voraussetzungen die Eigenschaften *sicher eingehalten* werden, insbesondere wird nicht erneut darauf hingewiesen, dass nachträgliche Änderungen Dritten gegenüber nicht nachweisbar sind. Wenn also davon die Rede ist, dass eine der Eigenschaften Append-Only, Consistent View oder totale Ordnung „erfüllt“ sei, dann ist damit gemeint, dass sie nicht verletzt wird – und eben nicht, dass eine Verletzung nachweisbar wäre.

[1] Zur Erinnerung: Wegen evtl. parallel eintreffenden Nachrichten kann es Uneinigkeit der Peers darüber geben, welche Nachricht zuerst eingetroffen ist. Siehe S. 46 (4.1.4) für eine ausführlichere Erklärung.

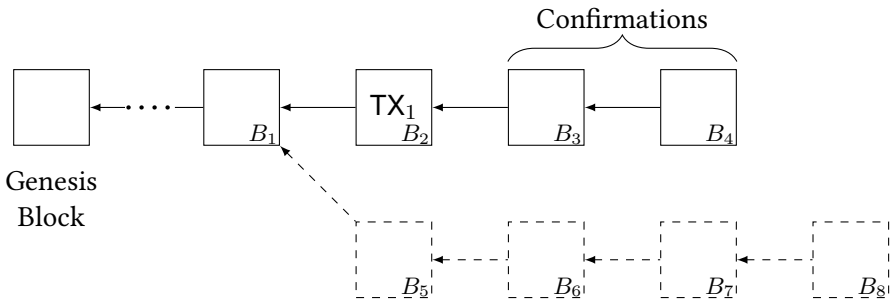


Abbildung 5.1: Nachträgliche Änderung trotz Confirmations. Zunächst ist B_1, B_2 der active Fork, in B_2 ist eine Transaktion TX_1 enthalten. Die Miner arbeiten zunächst weiter am B_1, B_2-Fork und erzeugen zwei Confirmations: B_3 und B_4. Dann werden die Blöcke B_5 bis B_8 veröffentlicht – die Teilnehmer übernehmen den neuen Fork als active Fork und TX_1 gilt wieder als offene Transaktion.

5.3 Blockchain als Bulletin Board

Eine erste – und naheliegende – Idee für ein Bulletin Board auf Basis einer Blockchain ist, die Blockchain direkt als Bulletin Board zu verwenden. Entsprechend wurde dieses Vorgehen auch schon mehrfach vorgeschlagen [26, 28, 76, 101, 110] bzw. wird von Unternehmen, die eVoting-Systeme anbieten, genutzt [41, 43] (allerdings mit permissioned Blockchains).

Um eine Nachricht auf einem solchen Bulletin Board zu veröffentlichen, verpackt der Absender seine Nachricht in eine Transaktion und schickt diese an einen oder mehrere Blockchain-Peers, wo sie wie jede andere Transaktion weitergeleitet und verarbeitet wird. Eine Nachricht gilt dann als veröffentlicht, sobald die entsprechende Transaktion in einen Block aufgenommen wird und dieser Block mindestens c Confirmations hat. Gelesen werden kann ein solches Bulletin Board dann über eine Full Node, welche aus den Transaktionen wieder die Nachrichten extrahiert.

Bevor die resultierenden Eigenschaften eines solchen Bulletin Boards untersucht werden, folgt zunächst eine kurze Vorstellung von Angriffsszenarien auf eine Blockchain, die vor allem dann interessant sind, wenn die Blockchain als Bulletin Board verwendet wird.

5.3.1 (Neue) Angriffsszenarien durch Verwendung einer Blockchain als Bulletin Board

Hier werden zwei Angriffsszenarien auf eine Blockchain vorgestellt, die vor allem dann relevant sind, wenn die Blockchain als Bulletin Board verwendet wird – sie sind aber prinzipiell auch ohne die Verwendung als Bulletin Board möglich. Daneben ist noch der in Abbildung 5.1 dargestellte Angriff auf die Persistenz-Eigenschaft möglich, der hier nicht nochmals wiederholt wird.

A.12 (Lebendigkeit) *Der Angreifer flutet die Blockchain mit mehr Transaktionen, als diese verarbeiten kann.*

Weil die Miner nur eine begrenzte Anzahl von Transaktionen pro Zeiteinheit in Blöcke des active Fork aufnehmen können (siehe Blockchain Trilemma, S. 32), kann ein Angreifer einfach mehr Transaktionen erzeugen und die Miner damit zwingen, nicht mehr alle offenen Transaktionen zu verarbeiten.

Dieser Angriff ist zunächst sehr ähnlich zu Angriff A.2 (Angreifer sendet sehr viele Nachrichten ans Bulletin Board). Falls die Blockchain – wie bei öffentlichen Blockchains üblich – jedoch eine digitale Währung implementiert, gibt es in der Regel Transaktionsgebühren und die Miner bevorzugen Transaktionen mit höheren Gebühren. Das kann sich ein Angreifer zunutze machen, indem er Transaktionen mit relativ hohen Transaktionsgebühren erzeugt. So kann er nicht nur einige zufällige, sondern potentiell *alle* legitimen Nachrichten an die Blockchain/das Bulletin Board „verdrängen".

A.13 (Lebendigkeit) *Die Miner können Transaktionen zensieren, indem sie diese ignorieren.*

Da die Miner auswählen, welche Transaktionen sie in einen Block aufnehmen, können sie dabei auch bestimmte Transaktionen ausschließen. So lange die meisten gültigen Blöcke von ehrlichen Minern erzeugt werden, wird eine vom Angreifer unerwünschte Nachricht zwar irgendwann in den active Fork aufgenommen, eine Verzögerung ist aber dennoch möglich. Allerdings hat der Angreifer bei Blockchains mit Block Rewards/Transaktionsgebühren hier einen „Hebel", um andere Miner ebenfalls zur Zensur zu bewegen: Der Angreifer kann unter den Minern bekannt machen, dass er keine Blöcke akzeptieren wird, die bestimmte Transaktionen beinhalten. Da die Miner ein Interesse haben, dass von ihnen erstellte Blöcke möglichst breite Akzeptanz finden (sonst bekommen sie keinen Block Reward), haben sie nun einen Anreiz am Angriff teilzunehmen.

Dieser Angriff ist sehr ähnlich zu Angriff A.11, hier ist aber für den Absender schwerer erkennbar, dass seine Nachricht ignoriert wird, weil eine gewisse Verzögerung zwischen Absenden einer Transaktion und Aufnahme in einen Block normal ist.

5.3.2 Diskussion zentraler Bulletin Board-Eigenschaften

In diesem Unterabschnitt wird für jede der zentralen Eigenschaften eines Bulletin Boards kurz diskutiert, inwiefern diese Eigenschaft von einem Bulletin Board, das aus einer Blockchain besteht, erfüllt werden kann. Dabei wird angenommen, dass ein Angreifer nicht in der Lage ist, das Kommunikationsnetzwerk zu kontrollieren – andernfalls wären Eclipse Attacks möglich (siehe Unterabschnitt 3.7.6).

Append-Only Da eine Nachricht als veröffentlicht gilt, sobald sie in einem Block mit c Confirmations enthalten ist, muss für die Erfüllung von Append-Only die Persistenz-Eigenschaft erfüllt sein. Wie in Abschnitt 3.8 beschrieben, kann das nur dann erreicht werden, wenn die meisten gültigen Blöcke von ehrlichen Minern erzeugt werden und die Nachrichtenverzögerung zwischen den Peers klein genug ist. Im Sprachgebrauch digitaler Währungen ausgedrückt: Double-Spending muss verhindert werden.

Außerdem müssen die Leser entweder Full Nodes betreiben oder zumindest viele der Full Nodes nach deren Sicht auf die Blockchain fragen, damit sie überhaupt sicher wissen können, welche Blöcke und Transaktionen mindestens die geforderten c Confirmations haben[2].

Receipts Receipts in der Form, wie sie in Abschnitt 4.1 gefordert werden, gibt es hier nicht – der Absender einer Transaktion bekommt zunächst keine direkte Antwort. Dem Absender bleibt also nichts anderes übrig, als darauf zu warten, dass seine Transaktion von einem Miner in einen Block aufgenommen wird – das überhaupt zu erkennen, erfordert auch von den Absendern den Betrieb einer Full Node bzw. das Anfragen bei vielen der vorhandenen Full Nodes. Sobald dieser Block dann c Confirmations erhalten hat, gilt die Nachricht als veröffentlicht. Diese Confirmations sind aber noch immer kein Receipt, weil sie nicht als Beweis dienen können, wenn die Nachricht am Ende nicht mehr Teil des Boardinhalts ist (siehe Abschnitt 5.2).

Consistent View Wie schon für Append-Only müssen die Leser auch für Consistent View Full Nodes betreiben oder bei vielen der bestehenden Full Nodes anfragen, um überhaupt erkennen zu können, wann eine Transaktion die nötigen c Confirmations erhalten hat. Die strenge Forderung aus der Definition von Consistent View, dass „sobald ein Leser eine Nachricht m auf dem Bulletin Board sieht, auch alle anderen Leser m sehen“, muss hier etwas aufgeweicht werden, um dem Umstand Rechnung zu tragen, dass es wegen Nachrichtenverzögerungen eine gewisse Zeit braucht, bis alle Full Nodes die c Confirmations „gesehen“ haben.

Zudem ist die Annahme, dass das Netzwerk nicht vom Angreifer kontrolliert wird, hier von entscheidender Bedeutung, sonst könnte der Angreifer mittels einer Eclipse Attack jedem Leser einen anderen active Fork präsentieren. Im Unterschied zu „klassischen“ verteilten Bulletin Boards würde durch eine Eclipse Attack nämlich nicht die Verfügbarkeit eingeschränkt, sondern Consistent View verletzt.

Auf der anderen Seite sind die Angriffe A.8 und A.9 nicht mehr möglich: Da der Angreifer den Lesern keine Blöcke vorenthalten kann, haben die Leser (nach einer gewissen Synchronisationszeit) immer dieselbe Sicht auf die Blockchain und damit das Bulletin Board.

Totale Ordnung Durch die feste Reihenfolge der Transaktionen innerhalb eines Blocks und die feste Reihenfolge der Blöcke ergibt sich automatisch eine totale Ordnung der Nachrichten. Diese Ordnung entspricht aber nicht dem Eingangszeitpunkt der Nachrichten, sondern wird von den Minern beliebig festgelegt. Das bedeutet, dass ein Angreifer Angriff A.12 und Angriff A.13 verwenden kann, um die resultierende Ordnung zu beeinflussen. Eine totale Ordnung auf Basis des Eingangszeitpunkts der Nachrichten ist also nicht möglich.

[2]Die beiden Alternativen sind bezüglich der Sicherheit sehr ähnlich: Wenn ein Großteil der vorhandenen Full Nodes „lügt“, dann könnten sie auf dieselbe Weise auch eine neue Full Node betrügen – denn eine neue Full Node bekommt ihre Informationen über die Vergangenheit (und zum Großteil auch Informationen über neue Blöcke und Transaktionen) von anderen Full Nodes.

Andere Möglichkeiten für eine totale Ordnung auf Basis des Inhalts der Nachrichten wie z. B. eine durch die Absender definierte Ordnung sind natürlich weiterhin möglich – allerdings muss berücksichtigt werden, dass die Nachrichten erst nach c Confirmations als veröffentlicht gelten und dass offene Transaktionen eine unbestimmte Zeit brauchen, bis sie in einen Block aufgenommen werden. Darum kann z. B. das Verfahren von Heather und Lundin, bei dem die Nachrichten einen Hashwert über den Boardinhalt *vor* dieser Nachricht enthalten, nicht ohne weiteres genutzt werden.

Zensurfreiheit Damit keine Nachrichten (egal ob gezielt oder nicht) zensiert werden, müssen in jedem Fall alle Transaktionen irgendwann in einen Block des active Fork aufgenommen werden. Für eine Blockchain, die Lebendigkeit erfüllt, ist das in endlicher Zeit per Definition der Fall. Bei einem Bulletin Board ist aber abhängig von der Anwendung auch Zeit ein wichtiger Faktor: Stimmabgaben für eine Wahl oder Gebote einer Auktion müssen innerhalb einer gewissen Zeit auf dem Bulletin Board erscheinen, um im Sinne der Anwendung noch gültig zu sein. Genau das könnte bei einer Blockchain, die eine Währung implementiert, aber mit dem in A.12 beschriebenen Fluten der Blockchain durch einen Angreifer verhindert werden.

Weiter muss auch Angriff A.13 verhindert werden. Folglich muss mehr als die Hälfte der gültigen Blöcke von ehrlichen Minern erzeugt werden *und* die Miner müssen irgendwie sicherstellen, dass alle Transaktionen rechtzeitig (im Sinne der Anwendung) in einen Block aufgenommen werden. Letzteres kann nur dann erreicht werden, wenn die Transaktionsrate der Blockchain höher als die Nachrichtenrate des Bulletin Boards ist und die Miner Bulletin Board-Transaktionen bevorzugt behandeln. Diese Bevorzugung könnte durch entsprechende Regeln der Blockchain gefördert werden, erfordert allerdings eine „angepasste“ Blockchain als Grundlage für das Bulletin Board– das widerspricht aber der oben gemachten Annahme, dass eine bestehende Blockchain verwendet wird.

Performance Die Anzahl an Nachrichten, die ein Bulletin Board, das aus einer Blockchain besteht, verarbeiten kann, hängt direkt mit der Transaktionsrate dieser Blockchain zusammen. Und die ist – wie in Unterabschnitt 3.7.1 und 3.7.5 behandelt – begrenzt und liegt bei den aktuell populärsten Blockchains im Bereich von wenigen Transaktionen pro Sekunde. Je nach Anwendung ist das mehrere Größenordnungen zu langsam für ein Bulletin Board: Wie auf S. 47 vorgerechnet, müssen z. B. für eine Bundestagswahl *im Mittel* ca. 1.700 Stimmabgaben pro Sekunde unterstützt werden.

Bei der Performance-Betrachtung muss auch der notwendige Aufwand zum Senden einer Nachricht und zum Lesen und ggf. Validieren des Bulletin Boards berücksichtigt werden. Da sowohl die Leser als auch die Absender von Nachrichten jeweils Full Nodes betreiben oder für jede Nachricht bei vielen Full Nodes anfragen müssen, ist dieser Aufwand verglichen mit klassischen Bulletin Boards sehr groß: Dort genügt es in der Regel, eine Signatur pro Nachricht oder sogar nur eine Signatur über den gesamten Boardinhalt zu prüfen. Ein Vorteil beim Betrieb einer Full Node durch einen Leser ist allerdings, dass alle notwendigen Daten lokal vorliegen,

d. h. die Performance eines einzelnen Lesevorgangs ist unabhängig von Netzwerkverzögerung und -bandbreite.

Verfügbarkeit Bei der Verfügbarkeit muss hier unterschieden werden zwischen Lese- und Schreibverfügbarkeit. Da die Leser ohnehin alle Blöcke und damit auch alle Nachrichten lokal als Teil ihrer Full Node vorhalten müssen, ist ein Lesezugriff selbst dann noch möglich, wenn alle anderen Peers ausfallen.

Um eine Nachricht zu veröffentlichen, braucht es (nur) einen erreichbaren Miner, der die Nachricht in einen Block aufnehmen kann. Dabei ist allerdings zu beachten, dass viele Konsens-Algorithmen beim plötzlichem Wegfall vieler Miner längere Zeit „stecken bleiben“, weil die verbleibenden Miner die bisherige Blockrate nicht aufrecht erhalten können[3]. Dennoch gilt für eine Blockchain, dass sie konzeptionell keinen Single Point of Failure hat und auch die erwartete Verfügbarkeit steigt mit der Anzahl der Peers (insbesondere Miner).

Analog zur Betrachtung bei der Zensurfreiheit muss aber auch hier die Möglichkeit von Angriff A.12 berücksichtigt werden, insbesondere die Variante mit Transaktionsgebühren kann dazu führen, dass das Bulletin Board nicht mehr zum Schreiben verfügbar ist.

5.3.3 Fazit

Eine Blockchain direkt als Bulletin Board zu verwenden, bringt mit Blick auf die von einem Bulletin Board geforderten Eigenschaften einige Probleme mit sich: Die zentrale Forderung nach Receipts kann nicht erfüllt werden und bei anderen zentralen Forderungen wie Append-Only und Consistent View ist eine Verletzung nicht zweifelsfrei nachweisbar und es kann maximal eine schwache Formulierung der Begriffe erfüllt werden. Zudem müssen Absender und Leser von Nachrichten relativ aufwändige Full Nodes betreiben und um überhaupt irgendwelche Garantien zu haben, muss die starke Annahme getroffen werden, dass der Angreifer nicht in der Lage ist, das Netzwerk zu kontrollieren[4].

Auf der positiven Seite ist eine größere öffentliche Blockchain – und damit auch ein darauf aufgebautes Bulletin Board– in der Praxis deutlich schwerer zu manipulieren als ein zentrales oder auch verteiltes Bulletin Board: Die Betreiber des oder der Bulletin Board-Peers werden in der Regel bestimmte Interessen bezüglich der konkreten Anwendung des Bulletin Boards verfolgen, insbesondere im Kontext von größeren Wahlen – sonst würden sie die Peers erst gar nicht betreiben. Bei einer weltweit verteilten Blockchain, deren Hauptanwendung eine andere – z. B. eine dezentrale Währung – ist, kann dagegen davon ausgegangen werden, dass ein Großteil der Miner kein Interesse daran hat, an einer Manipulation mitzuwirken, die das Vertrauen in die Hauptanwendung schmälern würde (und sich damit negativ auf das „Vermögen“ der

[3] Z. B. bei Proof of Work-Blockchains: Die Schwierigkeit des Proof of Work wird stark „geglättet“ angepasst, um Oszillationen zu verhindern. Analog auch für Proof of Capacity u. ä. Auch viele Ausprägungen von Proof of Stake haben dieses Problem, weil mit dem Wegfall fast aller Miner auch fast alle Slots für Blöcke unbesetzt bleiben.

[4] Bei einer Anwendung für nationale Wahlen ist das tatsächlich eine sehr starke Annahme: Selbst Regierungen in Demokratien nehmen immer wieder Einfluss auf die Nachrichtenübertragung im Internet (z. B. [58, 59]).

Miner auswirkt). Nun kann bei den meisten in Abschnitt 4.3 vorgestellten Bulletin Boards eine Manipulation zwar nachgewiesen werden, im Falle einer Wahl wäre der Vertrauensverlust bei Bekanntwerden einer Manipulation aber bereits ein immenser Schaden.

Ein weiterer Pluspunkt ist die inhärente Archivierung aller Nachrichten, wie sie von Hauser und Haenni [53] angesprochen wird: Bei einem klassischen Bulletin Board müsste entweder in Kauf genommen werden, dass das Bulletin Board irgendwann abgeschaltet wird – und damit im Prinzip Append-Only verletzt – oder das Bulletin Board bzw. ein anderes Archiv müssten nur für die Bulletin Board-Anwendung weiter betrieben werden. Eine öffentliche Blockchain dagegen wird – soweit sie eine Anwendung jenseits des Bulletin Boards hat – von den Peers aus anderer Motivation heraus weiter betrieben.

Daher wäre eine „gemischte“ Lösung wünschenswert, bei der die jeweiligen Vorteile genutzt werden können – hohe praktische Manipulationssicherheit und die Archivierungsfunktion einer Blockchain einerseits und die Nachweisbarkeit von Fehlverhalten eines verteilten Bulletin Boards andererseits.

5.4 Hybride Modelle aus klassischem Bulletin Board und Blockchain

Aus den Überlegungen in Unterabschnitt 5.3.3 heraus wird in diesem Abschnitt diskutiert, wie ein hybrides Bulletin Board aussehen könnte, bei dem sowohl Ideen aus „klassischen“ Bulletin Boards als auch Blockchains eingesetzt werden.

Dafür sind mehrere Modelle denkbar, die hier kurz aufgezählt und deren Eigenschaften im Folgenden untersucht werden. „Bulletin Board“ steht in dieser Aufzählung jeweils für ein klassisches (verteiltes) Bulletin Board:

M.1 Das Bulletin Board sendet zusätzlich zur internen Speicherung alle veröffentlichten Nachrichten signiert an eine Blockchain. Gelesen und geschrieben wird in der Regel direkt beim Bulletin Board, die Daten auf der Blockchain werden „nur“ durch Beobachter und im Streitfall verwendet.

M.2 Das Bulletin Board legt einen Hashwert über jede empfangene Nachricht signiert auf der Blockchain ab. Diese Lösung ist im Prinzip sehr ähnlich zu Modell M.1, die nötige Datenmenge auf der Blockchain wird jedoch deutlich reduziert. Zudem könnte das Bulletin Board zuerst den Hashwert auf der Blockchain ablegen und erst dann die zugehörige Nachricht veröffentlichen, so kann verhindert werden, dass die Miner gezielt Nachrichten zensieren können (Angriff A.13).

M.3 Wie Modell M.1, jedoch stellt das Bulletin Board keine expliziten Receipts mehr aus, sondern das signierte Ablegen einer Nachricht auf der Blockchain gilt als Receipt. Absender müssen dementsprechend sowohl mit dem Bulletin Board kommunizieren als auch an der Blockchain teilnehmen.

M.4 Das Bulletin Board sendet in regelmäßigen Abständen einen signierten Hashwert über den aktuellen Boardinhalt an eine Blockchain. Wie bei Modell M.1 werden die Daten auf der Blockchain hier nur von Beobachtern und im Streitfall verwendet.

All diesen Modellen ist zunächst gemein, dass die Blockchain „nur“ zum Protokollieren der Bulletin Board-Aktivitäten verwendet wird. Der Grund dafür wurde im letzten Abschnitt diskutiert: Die Blockchain in irgendeiner Form direkt als Bulletin Board zu verwenden, führt dazu, dass einige wichtige Eigenschaften aufgeweicht oder ganz aufgegeben werden müssen, wie z. B. die Receipts. Da keines der Modelle spezielle Eigenschaften der Blockchain fordert, können auch jeweils mehrere Blockchains oder sogar mehrere Modelle zugleich zum Einsatz kommen.

5.4.1 Untersuchung der Modelle

Im Folgenden werden die Modelle M.1 – M.4 genauer untersucht, insbesondere im Hinblick darauf, welche Vor- und Nachteile sie gegenüber einem rein „klassischen“ bzw. einer Blockchain als Bulletin Board (wie in Abschnitt 5.3 diskutiert) haben; dabei wird in der Regel informell argumentiert. Zunächst werden die allgemeingültigen Vor- und Nachteile – d. h. solche, die auf jedes Modell zutreffen – und anschließend Spezifika der einzelnen Modelle diskutiert. Für eine Blockchain gilt auch hier wieder, dass sie – soweit nicht anders verzeichnet – die Eigenschaften Persistenz und Lebendigkeit erfüllen soll.

Vor- und Nachteile für alle Modelle Da alle Modelle eine Art Protokoll der Bulletin Board-Aktivitäten auf einer Blockchain bieten, welches wegen der zunehmenden Anzahl an Confirmations „immer sicherer“ wird, erlauben sie alle *nachträgliche* Kontrollen des Bulletin Boards. Das bedeutet, dass z. B. Wahlbeobachter auch nach Ende einer Wahl noch tätig werden können und nicht zwingend *während* der Wahl immer wieder die Boardinhalte auslesen müssen.

Weiter enthalten alle Modelle die Möglichkeit, Receipts und signierte Lesevorgänge zu implementieren (bei Modell M.3 müssen die Absender die Signaturen dafür jeweils selbst zusammensetzen). Dadurch werden Verletzungen von Append-Only, Consistent View, Receipts und totaler Ordnung nachweisbar – mit einer Blockchain als Bulletin Board ist das nicht möglich.

Wie bereits in Unterabschnitt 5.3.3 ausgeführt, bietet eine öffentliche Blockchain, deren Hauptanwendung nicht das Bulletin Board ist, den Vorteil, dass dort abgelegte Transaktionen „automatisch“ archiviert sind. Für die Modelle M.1 und M.3 muss also außerhalb der Blockchain nichts mehr archiviert werden. Bei M.2 und M.4 genügt es, die Nachrichten ausfallsicher zu archivieren, die Integrität kann mithilfe der Hashwerte auf der Blockchain geprüft werden.

Zudem ist es in allen Modellen möglich, dass ein Leser nicht nur mit dem Bulletin Board kommuniziert, sondern zugleich an der entsprechenden Blockchain teilnimmt. Ein solcher Leser kann dann – ohne Kontakt mit anderen Lesern – Verletzungen von Consistent View erkennen und nachweisen. Das ist ein Vorteil gegenüber klassischen Bulletin Boards, bei denen die Leser sich dafür gegenseitig „außerhalb“ des Bulletin Boards finden müssen – setzt aber voraus, dass

die Leser eine Full Node betreiben oder zumindest bei den meisten der bestehenden Full Nodes die für sie relevanten Daten anfragen.

Spezifische Vor- und Nachteile Bei Modell M.1 wird die Idee einer Blockchain als „Protokoll“ des Bulletin Boards sehr direkt umgesetzt, indem alle veröffentlichten Nachrichten auf die Blockchain geschrieben werden. Folglich gibt es hier dieselben Probleme mit der Transaktionsrate der Blockchain wie sie in Abschnitt 5.3 beschrieben wurden.

Schon in der kurzen Beschreibung zu Modell M.2 erwähnt ist der Vorteil, dass die Datenmenge, die auf die Blockchain geschrieben werden muss, deutlich kleiner ist als bei direkter Verwendung der Blockchain als Bulletin Board. Das erlaubt zum einen einen höheren Durchsatz – die Blockgröße wird in der Regel durch eine Anzahl Bytes, nicht durch eine Anzahl Transaktionen beschränkt – und zum anderen macht es die Transaktionen, die dafür nötig sind, günstiger[5]. Dennoch muss auch hier für jede veröffentlichte Nachricht eine Transaktion an die Blockchain gesendet werden, die wiederum eine begrenzte Transaktionsrate hat.

Im Modell M.3 muss auch wieder jede einzelne Nachricht in eine Transaktion verpackt und auf die Blockchain geschrieben werden, entsprechend ist der Durchsatz durch die Transaktionsrate der Blockchain beschränkt. Zudem brauchen Absender und Leser von Nachrichten jeweils eine Full Node bzw. müssen bei vielen Full Nodes anfragen. Dafür ist es für das Bulletin Board deutlich „riskanter“, ein Receipt auszustellen, die Nachricht aber nicht jedem Leser anzuzeigen: Falls der Leser auch die Receipts auf der Blockchain liest, könnte er diesen Betrug ohne Mitwirkung des Absenders nachweisen.

Modell M.4 bietet gegenüber den anderen Modellen und einer Blockchain als Bulletin Board den Vorteil, dass die Transaktionsrate der Blockchain zunächst keinen Einfluss auf den maximalen Durchsatz des Bulletin Boards hat. Dazu kommt, dass die einzelnen Transaktionen relativ wenige Daten enthalten und entsprechend billig sind. Allerdings ist der nötige Aufwand zum Validieren des Boardinhalts anhand der Daten auf der Blockchain deutlich höher: Es kann immer nur der komplette Boardinhalt auf einmal validiert werden. Insbesondere für Absender, die sich „vergewissern“ wollen, dass ihre Nachricht in den Hashwerten, die das Bulletin Board auf der Blockchain ablegt, enthalten ist, bedeutet das einen erheblichen Mehraufwand.

5.4.2 Fazit

Zusammenfassend lässt sich festhalten, dass eine Blockchain direkt als Bulletin Board zu verwenden, eher Nach- als Vorteile gegenüber einem klassischen Bulletin Board bringt. Insbesondere die stark begrenzte Transaktionsrate aktueller Blockchains stellt hier ein Problem dar, falls jede einzelne Nachricht, die auf dem Bulletin Board veröffentlicht werden soll, zu einer Transaktion führt.

[5] Da die Blockgröße in der Regel durch eine Anzahl Bytes beschränkt ist, bevorzugen die Miner natürlich jene Transaktionen, bei denen die Transaktionsgebühr pro Byte am höchsten ist. Dementsprechend sind „kleinere“ Transaktionen billiger.

Aus diesem Grund ist Modell M.4 das für zukünftige Untersuchungen vielversprechendste Modell: Damit sind sowohl wie oben beschrieben nachträgliche Kontrollen des Bulletin Boards möglich, als auch die Anforderungen an Nachweisbarkeit von Veränderungen erfüllt und zugleich kann eine sehr hohe Performance erreicht werden.

5.5 Smart Contract als Bulletin Board

Eine weitere Möglichkeit, ein Bulletin Board auf Basis einer Blockchain zu implementieren, ist die Verwendung eines *Smart Contracts*. Ein Smart Contract ist ein Programm, das „auf einer Blockchain ausgeführt wird“ – im Normalfall erzeugt und gestartet durch Transaktionen auf der Blockchain[6]. Das Besondere gegenüber „normalen“ Programmen ist, dass die Ausführung von allen am Konsens-Algorithmus teilnehmenden Parteien verifiziert wird – solange die Blockchain selbst „sicher“ ist, wird das Programm also öffentlich überprüfbar korrekt ausgeführt. Genau diese Eigenschaft macht einen Smart Contract auf den ersten Blick attraktiv, um ein Bulletin Board zu implementieren.

Inwiefern diese Einschätzung einem zweiten Blick standhält, wird nach einer kurzen Einführung zu Smart Contracts in Ethereum diskutiert.

5.5.1 Einführung zu Smart Contracts in Ethereum

Die aktuell bekannteste öffentliche Blockchain, die explizit für Smart Contracts entwickelt wird, ist Ethereum [37, 106]. Neben den Smart Contracts gibt es in Ethereum auch eine digitale Währung namens *Ether*. Ein Smart Contract in Ethereum ist eine Sequenz aus Anweisungen – also ein Programm – für die *Ethereum Virtual Machine* (EVM). Erzeugt wird ein Smart Contract durch eine Transaktion, die diese Sequenz aus Anweisungen enthält. Durch diese Transaktion wird auch eine „Adresse“ für den Smart Contract erzeugt, unter welcher dieser anschließend erreichbar ist.

Um das Programm zu starten, bedarf es einer weiteren Transaktion an die Adresse des Smart Contracts, üblicherweise enthält diese Transaktion einen Funktionsnamen und Parameter, mit denen die entsprechende Funktion dann aufgerufen wird[7]. Wirklich ausgeführt wird das Programm erst in dem Moment, in dem ein Miner die aufrufende Transaktion in einen Block aufnimmt. Damit ein Miner die zur Ausführung notwendige Rechenleistung überhaupt investiert, gibt es für jede Anweisung der EVM einen Preis, welcher in der Einheit gas gemessen wird.

[6] Es ist z. B. in Ethereum auch möglich, dass ein Smart Contract einen anderen aufruft. Zudem es gibt sogenannte *precompiled contracts*, die nicht auf der Blockchain liegen, sondern direkt Teil der Client-Software sind. Prinzipiell könnte man diese precompiled contracts aber auch als zusätzliche Op-Codes der EVM (siehe nächsten Abschnitt) betrachten.

[7] Tatsächlich kann die Transaktion beliebige Daten enthalten, die dann an den Smart Contract übergeben werden – in diesem Fall wird die sogenannte *fallback function* des Smart Contract aufgerufen. Falls die Daten nicht zu dem passen, was der Smart Contract erwartet, können die Ergebnisse entsprechend unerwartet sein – der hier beschriebene Fall ist lediglich der „normale“ Funktionsaufruf. Außerdem ist ein Aufruf wie oben erwähnt auch durch einen anderen Smart Contract möglich.

Der Absender der aufrufenden Transaktion legt dabei einen *Gas Price* fest, also die Menge an *Ether*, die er pro fälligem gas bereit ist, an den Miner zu bezahlen.

Da die Menge an „verbrauchtem“ gas entscheidend dafür ist, wie aufwändig es ist, einen Block zu verifizieren, sind Ethereum-Blöcke nicht durch eine bestimmte Anzahl Bytes oder Transaktionen, sondern durch das *Block Gas Limit* – also die maximale Menge gas, die in einem Block „verbraucht“ werden darf – in ihrer Größe beschränkt[8].

Damit alle Peers der Blockchain zum gleichen Ergebnis für eine Programmausführung kommen, gibt es einige wichtige Einschränkungen für Smart Contracts: Erstens müssen sie komplett deterministisch sein, d. h. es gibt keinen Zugriff auf Zufallsbits (es sei denn, diese werden durch eine Transaktion an den Smart Contract übergeben – der Smart Contract selbst ist dann aber trotzdem noch deterministisch).

Zweitens muss das Programm terminieren. Das wird in Ethereum zum einen dadurch erzwungen, dass eine Transaktion natürlich nicht mehr gas als das Block Gas Limit verbrauchen kann. Zum anderen wird jeder Transaktion eine vom Absender bestimmte Menge Ether beigefügt, um das verbrauchte gas gemäß dem gewählten Gas Price zu bezahlen – ist diese Menge Ether verbraucht, wird die Programmausführung abgebrochen.

Und drittens ist die Ausführung komplett von der „Außenwelt“ isoliert, d. h. alle für die Ausführung notwendigen Daten müssen im Code bzw. internen Zustand des Smart Contracts vorliegen. Das wiederum schränkt den Gebrauch von Kryptographie sehr stark ein, weil z. B. kein privater Schlüssel in einem Smart Contract gespeichert werden kann – damit wäre er für alle Teilnehmer der Blockchain lesbar und somit eben nicht mehr *privat*.

5.5.2 Smart Contract als Bulletin Board

Für einen Smart Contract, der ein Bulletin Board implementiert, kann gleich zu Beginn festgehalten werden, dass es keine explizite Funktion zum Lesen der Boardinhalte braucht: Da ein Leser ohnehin eine Full Node betreiben oder die meisten bestehenden Full Nodes anfragen muss, kann er auch direkt den internen Zustand des Smart Contracts – und damit die Boardinhalte – lesen bzw. anfragen.

Folglich braucht ein solcher Smart Contract lediglich eine Funktion, um Nachrichten zu veröffentlichen. Auch hier kann festgehalten werden: Diese Funktion kann kein Receipt in Form einer Signatur an den Absender zurück liefern, weil der Smart Contract wie oben beschrieben keinen privaten Schlüssel enthalten kann. Weil der Smart Contract aber ohnehin nur innerhalb der Blockchain „existiert“, können prinzipiell keine Garantien gegeben werden, welche über die der Blockchain selbst hinausgehen.

[8] Da jede Transaktion und jedes Byte einer Transaktion ebenfalls mit einem gewissen Preis in gas behaftet sind, gibt es indirekt durchaus eine Obergrenze für die Anzahl der Transaktionen und Bytes in einem Ethereum-Block [106, S. 20] – die entscheidende Größe ist aber das Block Gas Limit.

Eine solche Funktion zum Veröffentlichen einer Nachricht muss wie oben beschrieben vom Absender der Nachricht mittels einer Transaktion aufgerufen werden, welche die zu veröffentlichende Nachricht enthält. Da durch seinen Code die Regeln, nach denen ein Smart Contract eine Nachricht ggf. ablehnen könnte, öffentlich sind und auch die eben erwähnte Transaktion mit der zu veröffentlichenden Nachricht zum Aufruf des Smart Contracts öffentlich sein muss, wird der Smart Contract an sich konzeptionell unnötig: Ein Leser könnte auch einfach direkt die Transaktionen lesen und – außerhalb der Blockchain – Filterregeln anwenden, um gültige von ungültigen Nachrichten zu trennen. Daran ändert sich auch nichts, wenn die Nachrichten an den Bulletin Board-Smart Contract verschlüsselt werden: Alle zum Entschlüsseln notwendigen Informationen müssen bereits im Smart Contract enthalten und damit öffentlich sein.

Im Ergebnis gibt es durch den Smart Contract also zunächst keinen Gewinn an Funktion oder Sicherheit gegenüber einer Blockchain als Bulletin Board, wie es in Abschnitt 5.3 diskutiert wurde. An dem im nächsten Unterabschnitt vorgestellten Beispiel wird jedoch deutlich, dass Smart Contracts dennoch eine sinnvolle Wahl für eine Bulletin Board-Anwendung sein können – wenn damit eben nicht *nur* ein Bulletin Board implementiert werden soll.

5.5.3 Smart Contract als komplettes eVoting-System

Ein „normales“ Bulletin Board mit einem Smart Contract zu implementieren, bringt – wie im letzten Unterabschnitt dargelegt – keinen entscheidenden Vorteil gegenüber der Verwendung einer Blockchain als Bulletin Board. Das ändert sich jedoch, wenn das Bulletin Board zusätzlich noch Aufgaben der Anwendung – z. B. eines eVoting-Systems – übernehmen soll. Dann kann ein Smart Contract z. B. zur Stimmabgabe motivieren, indem jedem Wähler ein gewisser Betrag überwiesen wird – oder Nichtwähler sanktionieren, indem sich Wähler zunächst mit einem Pfand registrieren müssen, das sie erst nach erfolgter Stimmabgabe zurück erhalten.

Einen Schritt weiter gehen dabei McCorry et al.: Sie haben ein ganzes eVoting-System in einem Ethereum Smart Contract implementiert [79]. Dabei geben die Wähler ihre Stimme verschlüsselt beim Smart Contract ab – den Schlüssel sk dazu kennt nur der jeweilige Wähler. Durch das dafür verwendete homomorphe Verschlüsselungsschema[9] ist es möglich, dass diese Stimmen in verschlüsselter Form addiert werden. Aus diesem verschlüsselten Ergebnis und ihren geheim gehaltenen Schlüsseln können die Wähler dann jeweils eine Art Teilschlüssel[10] für das Gesamtergebnis berechnen – dieser Teilschlüssel gibt aber keinerlei Informationen über sk preis. Die Teilschlüssel wiederum werden an den Smart Contract gesendet, der daraus den „Gesamtschlüssel“ berechnet, mit dem das Ergebnis der Wahl entschlüsselt werden kann[11].

[9] Eine homomorphe Verschlüsselung erlaubt es, auf den verschlüsselten Daten bestimmte Operationen auszuführen, ohne sie dabei zu entschlüsseln – so kann z. B. die Summe einiger Zahlen gebildet werden, ohne dass dem „Rechner“ die Eingangsgrößen oder das Ergebnis bekannt werden. Bekannte teilhomomorphe (d. h. die möglichen Funktionen sind begrenzt) Verfahren sind z. B. von ElGamal [35], Benaloh [10] oder Paillier [89]; neuere Entwicklungen für vollhomomorphe Verfahren stammen z. B. von Gentry [46] und Brakerski [17].

[10] Das Wort „Teilschlüssel“ erleichtert das Verständnis, bedarf aber einer Klarstellung: Nur *alle Teilschlüssel zusammen* können zu einem Gesamtschlüssel kombiniert werden.

[11] Tatsächlich wird das Ergebnis nicht direkt entschlüsselt, sondern der „Gesamtschlüssel“ ermöglicht das „Knacken“ eines diskreten Logarithmus.

Da die notwendigen Datenmengen und Berechnungen vergleichsweise aufwändig sind, ist dieses eVoting-System jedoch stark beschränkt, was die Anzahl der Wähler angeht: Pro Ethereum-Block kann wegen des Block Gas Limits nur eine Stimmabgabe verarbeitet werden und beim Entschlüsseln des Endergebnisses wird dieses schon mit wenigen Dutzend Wählern erreicht.

6 Zusammenfassung und Ausblick

Auf Basis der vorgestellten Blockchain-Technologie und einer ausführlichen Diskussion von Bulletin Boards und ihren Eigenschaften wurde grundsätzlich untersucht, inwiefern die Entwicklung eines Bulletin Boards durch den Einsatz von Blockchains vereinfacht oder verbessert werden könnte. Motiviert war diese Untersuchung durch die augenscheinliche Verwandtschaft der wichtigsten Eigenschaften von Blockchains und Bulletin Boards wie z. B. die Unveränderbarkeit einmal aufgenommener Inhalte.

Dabei hat sich herausgestellt, dass bei großen öffentlichen Blockchains die für ein Bulletin Board geforderten Eigenschaften zwar schwerer zu verletzen sind, diese Verletzungen dann aber nicht – wie für Bulletin Boards gefordert – nachweisbar sind. Außerdem folgt aus dem Blockchain Trilemma, dass eine sichere und dezentrale Blockchain keinen beliebig hohen Durchsatz erzielen kann – die aktuell populären öffentlichen Blockchains sind z. B. mehrere Größenordnungen zu langsam, als dass sie ein Bulletin Board für eine Bundestagswahl darstellen könnten.

Dementsprechend wurden auch hybride Modelle untersucht, die in einen „klassischen“ Bulletin Board-Aufbau eine Blockchain mit einbinden, um möglichst die positiven Eigenschaften beider „Welten“ zu vereinen. Als am vielversprechendsten stellte sich dabei ein Modell heraus, in dem das Bulletin Board in regelmäßigen Abständen einen signierten Hashwert über den aktuellen Boardinhalt an die Blockchain sendet.

Ebenfalls untersucht wurde die Möglichkeit, ein Bulletin Board in einem Smart Contract zu implementieren. Weil Code und interner Zustand eines Smart Contracts inhärent öffentlich sind, ergab diese Untersuchung, dass ein als Smart Contract implementiertes Bulletin Board keinen signifikanten Vorteil gegenüber der direkten Verwendung einer Blockchain als Bulletin Board bietet.

Ausgehend von dem Modell, bei dem das Bulletin Board in regelmäßigen Abständen Hashwerte über seinen Inhalt an die Blockchain sendet, könnten sich zukünftige Arbeiten damit beschäftigen, welchen Einfluss auf die Eigenschaften des Bulletin Boards die Wahl der Parameter hat: Also welche konkrete Blockchain – insbesondere der verwendete Konsens-Algorithmus – und welche Zeitintervalle gewählt werden. Eine weitere offene Frage ist, ob es neben den identifizierten Dimensionen *Nachweisbarkeit von* und *Resistenz gegen Veränderungen* noch weitere gibt, in denen sich Blockchains und Bulletin Boards– bezüglich der für ein Bulletin Board notwendigen Eigenschaften – grundlegend unterscheiden. Auch die konkrete Implementierung eines solchen Bulletin Boards ist eine Aufgabe für zukünftige Arbeiten.

Literatur

[1] Ben Adida. „Advances in Cryptographic Voting Systems“. Diss. MIT Department of Electrical Engineering und Computer Science, 31. Aug. 2006.

[2] Ben Adida. „Helios: Web-based Open-Audit Voting“. In: *USENIX Security Symposium (Proceedings)*. USENIX SS 2008. (San Jose, CA, USA, 28. Juni 2008). USENIX Association, 2008, S. 335–348.

[3] Ben Adida, Olivier de Marneffe, Olivier Pereira und Jean-Jacques Quisquater. „Electing a University President Using Open-Audit Voting: Analysis of Real-World Use of Helios“. In: *Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (Proceedings)*. EVT/WOTE 2009. (Montreal, Kanada, 10. Aug. 2009). USENIX Association, 2009, S. 1–15.

[4] Ben Adida und Ronald L. Rivest. „Scratch & Vote“. In: *ACM Workshop on Privacy in the Electronic Society (Proceedings)*. WPES 2006. (Alexandria, VA, USA, 30. Okt. 2006). ACM, 2006, S. 29–40.

[5] Roberto Araújo, Sébastien Foulle und Jacques Traoré. „A Practical and Secure Coercion-Resistant Scheme for Internet Voting (Extended Abstract)“. In: *Towards Trustworthy Elections. New Directions in Electronic Voting*. Hrsg. von David Chaum, Markus Jakobsson, Ronald L. Rivest, Peter Y. A. Ryan, Josh Benaloh, Miroslaw Kutylowski und Ben Adida. 1. Aufl. Lecture Notes in Computer Science 6000. Springer Berlin Heidelberg, 2010, S. 330–342. ISBN: 978-3-642-12980-3.

[6] Ahmed Ben Ayed. „A Conceptual Secure Blockchain Based Electronic Voting System“. In: *International Journal of Network Security & Its Applications* 9.3 (Mai 2017), S. 1–9.

[7] Adam Back. *Hashcash Website*. 1997. URL: `http://www.hashcash.org/` (besucht am 19. 01. 2018).

[8] Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern und Guillaume Poupard. „Practical Multi-candidate Election System“. In: *ACM Symposium on Principles of Distributed Computing (Proceedings)*. PODC 2001. (Newport, RI, USA, 26. Aug. 2001). ACM, 2001, S. 274–283.

[9] Susan Bell, Josh Benaloh, Michael D. Byrne, Dana DeBeauvoir, Bryce Eakin, Gail Fisher, Philip Kortum, Neal McBurnett, Julian Montoya und Michelle Parker. „Star-Vote: A Secure, Transparent, Auditable, and Reliable Voting System“. In: *USENIX Journal of Election Technology and Systems* 1.1 (Aug. 2013), S. 18–37. ISSN: 2328-2797.

[10] Josh Benaloh. „Dense Probabilistic Encryption“. In: *Selected Areas in Cryptography (Proceedings)*. SAC 1994. (Kingston, Kanada). Aug. 1994.

[11] Josh Cohen Benaloh und Moti Yung. „Distributing the Power of a Government to Enhance the Privacy of Voters (Extended Abstract)“. In: *ACM Symposium on Principles of Distributed Computing (Proceedings)*. PODC 1986. (Calgary, Kanada, 11. Aug. 1986). ACM, 1986, S. 52–62.

[12] Josh Benaloh und Dwight Tuinstra. „Receipt-free Secret-ballot Elections (Extended Abstract)“. In: *ACM Symposium on Theory of Computing (Proceedings)*. STOC 1994. (New York, NY, USA, 23. Mai 1994). ACM, 1994, S. 544–553.

[13] Guido Bertoni, Joan Daemen, Michael Peeters, Gilles Van Assche und Ronny Van Keer. *Team Keccak Website*. 2018. URL: `https://keccak.team/index.html` (besucht am 26. 03. 2018).

[14] Albrecht Beutelspacher, Jörg Schwenk und Klaus-Dieter Wolfenstetter. *Moderne Verfahren der Kryptographie*. 8. Aufl. Vieweg+Teubner Verlag, 11. Aug. 2015. 186 S. ISBN: 978-3-8348-1927-7.

[15] BitcoinWisdom.com. *Bitcoin Difficulty*. 2018. URL: `https://bitcoinwisdom.com/bitcoin/difficulty` (besucht am 25. 04. 2018).

[16] Dan Boneh und Philippe Golle. „Almost Entirely Correct Mixing with Applications to Voting“. In: *ACM Conference on Computer and Communications Security (Proceedings)*. CCS 2002. (Washington, DC, USA, 18. Nov. 2002). ACM, 2002, S. 68–77.

[17] Zvika Brakerski. „Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP“. In: *Advances in Cryptology (Proceedings)*. CRYPTO 2012. (Santa Barbara, CA, USA, 19. Aug. 2012). Hrsg. von Reihaneh Safavi-Naini und Ran Canetti. Lecture Notes in Computer Science 7417. Springer Berlin Heidelberg, 2012, S. 868–886.

[18] Craig Burton, Chris Culnane und Steve Schneider. „vVote: Verifiable Electronic Voting in Practice“. In: *IEEE Security & Privacy* 14.4 (Juli 2016), S. 64–73. ISSN: 1540-7993.

[19] Vitalik Buterin. *Ethereum Sharding FAQ*. Version e46274ee...fb15ba85. 28. Nov. 2016. URL: `https://github.com/ethereum/wiki/wiki/Sharding-FAQ` (besucht am 25. 04. 2018).

[20] Richard Carback, David Chaum, Jeremy Clark, John Conway, Aleksander Essex, Paul S. Herrnson, Travis Mayberry, Stefan Popoveniuc, Ronald L. Rivest, Emily Shen, Alan T. Sherman und Poorvi L. Vora. „Scantegrity II Municipal Election at Takoma Park: The First E2E Binding Governmental Election with Ballot Privacy“. In: *USENIX Security Symposium (Proceedings)*. USENIX SS 2010. (Washington, DC, USA, 11. Aug. 2010). USENIX Association, 2010, S. 1–16.

[21] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Peter Y. A. Ryan, Emily Shen und Alan T. Sherman. „Scantegrity II: End-to-End Verifiability for Optical Scan Election Systems Using Invisible Ink Confirmation Codes“. In: *Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (Proceedings)*. EVT/WOTE 2008. (Washington, DC, USA, 28. Juli 2008). USENIX Association, 2008, S. 1–13.

[22] David Chaum, Debajyoti Das, Farid Javani, Aniket Kate, Anna Krasnova, Joeri De Ruiter und Alan T. Sherman. „cMix: Mixing with Minimal Real-Time Asymmetric Cryptographic Operations“. In: *Applied Cryptography and Network Security*. Springer International Publishing, 2017, S. 557–578.

[23] David Chaum und Eugène van Heyst. „Group Signatures“. In: *Advances in Cryptology (Proceedings)*. EUROCRYPT 1991. (Brighton, England, 8. Apr. 1991). Hrsg. von Donald W. Davies. Lecture Notes in Computer Science 547. Springer Berlin Heidelberg, 1991, S. 257–265.

[24] David Chaum, Peter Y. A. Ryan und Steve Schneider. „A Practical Voter-Verifiable Election Scheme“. In: *European Symposium on Research in Computer Security (Proceedings)*. ESORICS 2005. (Milan, Italien, 12. Sep. 2005). Hrsg. von Sabrina de Capitani di Vimercati, Paul Syverson und Dieter Gollmann. Lecture Notes in Computer Science 3679. Springer Berlin Heidelberg, 2005, S. 118–139.

[25] Nikos Chondros, Bingsheng Zhang, Thomas Zacharias, Panos Diamantopoulos, Stathis Maneas, Christos Patsonakis, Alex Delis, Aggelos Kiayias und Mema Roussopoulos. „D-DEMOS: A Distributed, End-to-End Verifiable, Internet Voting System“. In: *IEEE International Conference on Distributed Computing Systems (Proceedings)*. ICDCS 2016. (Nara, Japan, 27. Juni 2016). IEEE, Juni 2016, S. 711–720. ISBN: 978-1-5090-1484-2.

[26] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk und Ian Miers. „Fairness in an Unfair World: Fair Multiparty Computation from Public Bulletin Boards“. In: *ACM Conference on Computer and Communications Security (Proceedings)*. CCS 2017. (Dallas, TX, USA, 30. Okt. 2017). ACM, 30. Okt. 2017, S. 719–728.

[27] Ronald Cramer, Rosario Gennaro und Berry Schoenmakers. „A Secure and Optimally Efficient Multi-Authority Election Scheme“. In: *European Transactions on Telecommunications* 8.5 (Sep. 1997), S. 481–490.

[28] Jason Paul Cruz und Yuichi Kaji. „E-voting System Based on the Bitcoin Protocol and Blind Signatures“. In: *IPSJ Transactions on Mathematical Modeling and Its Applications* 10.1 (20. März 2017), S. 14–22. ISSN: 1882-7780.

[29] Chris Culnane, Peter Y. A. Ryan, Steve Schneider und Vanessa Teague. „vVote: A Verifiable Voting System“. In: *ACM Transactions on Information and System Security* 18.1 (Juni 2015). Hrsg. von Gene Tsudik, 3:1–3:30.

[30] Chris Culnane und Steve Schneider. „A Peered Bulletin Board for Robust Use in Verifiable Voting Systems“. In: *IEEE Computer Security Foundations Symposium (Proceedings)*. CSF 2014. (Wien, Österreich, 19. Juli 2017). IEEE, 20. Nov. 2014, S. 169–183. ISBN: 978-1-4799-4290-9.

[31] Whitfield Diffie und Martin Hellman. „New directions in cryptography“. In: *IEEE Transactions on Information Theory* 22.6 (Nov. 1976), S. 644–654.

[32] Danny Dolev und Andrew C. Yao. „On the Security of Public Key Protocols“. In: *IEEE Transactions on Information Theory* 29.2 (März 1983), S. 198–208.

[33] Jean Dollimore, Tim Kindberg und George Coulouris. *Distributed Systems: Concepts and Design*. Hrsg. von Andrew D. McGettrick. 4. Aufl. Addison Wesley, 2005. 944 S. ISBN: 978-0-321-26354-4.

[34] Cynthia Dwork und Moni Naor. „Pricing via Processing or Combatting Junk Mail“. In: *Advances in Cryptology (Proceedings)*. CRYPTO 1992. (Santa Barbara, CA, USA, 16. Aug. 1992). Hrsg. von Ernest F. Brickell. Lecture Notes in Computer Science 740. Springer Berlin Heidelberg, 1992, S. 139–147.

[35] Taher Elgamal. „A public key cryptosystem and a signature scheme based on discrete logarithms“. In: *IEEE Transactions on Information Theory* 31.4 (Juli 1985), S. 469–472.

[36] Wolfgang Ertel. *Angewandte Kryptographie*. 4. Aufl. Hanser Fachbuchverlag, 2007. 224 S. ISBN: 978-3-446-41195-1.

[37] Ethereum Community. *Ethereum Whitepaper. A Next-Generation Smart Contract and Decentralized Application Platform*. Version b0b035...ba63a2. 18. Sep. 2017. URL: `https://github.com/ethereum/wiki/wiki/White-Paper` (besucht am 26. 10. 2017).

[38] Ethereum Community. *Proof of Stake FAQ*. 31. Okt. 2017. URL: `https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ` (besucht am 14. 12. 2017).

[39] Ariel J. Feldman, J. Alex Halderman und Edward W. Felten. „Security Analysis of the Diebold AccuVote-TS Voting Machine“. In: *Electronic Voting Technology Workshop (Proceedings)*. EVT 2007. (Boston, MA, USA, 6. Aug. 2007). USENIX Association, 2007, S. 1–16.

[40] Michael J. Fischer und Nancy A. Lynch. „A lower bound for the time to assure interactive consistency“. In: *Information Processing Letters* 14.4 (Juni 1982), S. 183–186.

[41] Follow My Vote INC. *Follow my Vote*. 2017. URL: `https://followmyvote.com/` (besucht am 31. 10. 2017).

[42] Atsushi Fujioka, Tatsuaki Okamoto und Kazuo Ohta. „A Practical Secret Voting Scheme for Large Scale Elections“. In: *Advances in Cryptology (Proceedings)*. AUSCRYPT 1992. (Queensland, Australien, 13. Dez. 1992). Hrsg. von Jennifer Seberry und Yuliang Zheng. Lecture Notes in Computer Science 718. Springer Berlin Heidelberg, 1993, S. 244–251.

[43] Leonardo Gammar, Jaron Lukasiewicz und Bryan Ford. *Agora Whitepaper. Bringing our voting systems into the 21st century*. Version v0.2. 2018. URL: `https://agora.vote/Agora_Whitepaper_v0.2.pdf` (besucht am 05. 04. 2018).

[44] Juan A. Garay, Aggelos Kiayias und Nikos Leonardos. „The Bitcoin Backbone Protocol: Analysis and Applications“. In: *Advances in Cryptology (Proceedings)*. EUROCRYPT 2015. (Sofia, Bulgarien, 26. Apr. 2015). Hrsg. von Elisabeth Oswald und Marc Fischlin. Lecture Notes in Computer Science 9057. Springer Berlin Heidelberg, 15. Apr. 2015, S. 281–310. ISBN: 978-3-662-46802-9.

[45] Sean Gauld, Franz von Ancoina und Robert Stadler. *The Burst Dymaxion. An Arbitrary Scalable, Energy Efficient and Anonymous Transaction Network Based on Colored Tangles*. Version 1.00. 27. Okt. 2017. URL: `http://www.burst-coin.org/wp-content/uploads/2017/07/The-Burst-Dymaxion-1.00.pdf` (besucht am 19. 01. 2018).

[46] Craig Gentry und Shai Halevi. „Implementing Gentry's Fully-Homomorphic Encryption Scheme". In: *Advances in Cryptology (Proceedings)*. EUROCRYPT 2011. (Tallinn, Estland, 15. Mai 2011). Hrsg. von Kenneth G. Paterson. Lecture Notes in Computer Science 6632. Springer Berlin Heidelberg, 2011, S. 129–148.

[47] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf und Srdjan Capkun. „On the Security and Performance of Proof of Work Blockchains". In: *ACM Conference on Computer and Communications Security (Proceedings)*. CCS 2016. (Wien, Österreich, 24. Okt. 2016). ACM, 2016, S. 3–16. ISBN: 978-1-4503-4139-4.

[48] Seth Gilbert und Nancy Lynch. „Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services". In: *ACM SIGACT News* 33.2 (Juni 2002), S. 51–59.

[49] Rop Gonggrijp und Willem-Jan Hengeveld. „Studying the Nedap/Groenendaal ES3B Voting Computer: A Computer Security Perspective". In: *Electronic Voting Technology Workshop (Proceedings)*. EVT 2007. (Boston, MA, USA, 6. Aug. 2007). USENIX Association, 2007, S. 1–16.

[50] Stuart Haber, Josh Benaloh und Shai Halevi. *The Helios e-Voting Demo for the IACR*. Techn. Ber. International Association for Cryptologic Research, 24. Mai 2010. 7 S.

[51] Stuart Haber und W. Scott Stornetta. „How to Time-Stamp a Digital Document". In: *Advances in Cryptology (Proceedings)*. CRYPTO 1990. (Santa Barbara, CA, USA, 11. Aug. 1990). Hrsg. von Alfred J. Menezes und Scott A. Vanstone. Lecture Notes in Computer Science 537. Springer Berlin Heidelberg, 1990, S. 437–455.

[52] Louise Hagström und Olivia Dahlquist. „Scaling Blockchain for the Energy Sector". Masterarbeit. Teknisk-naturvetenskaplig fakultet, Uppsala Universitet, Juni 2017. 73 S.

[53] Severin Hauser und Rolf Haenni. „A Generic Interface for the Public Bulletin Board Used in UniVote". In: *Conference for E-Democracy and Open Government (Proceedings)*. CeDEM 2016. (Krems, Österreich, 18. Mai 2016). IEEE, Mai 2016, S. 49–56. ISBN: 978-1-5090-1042-4.

[54] Adam S. Hayes. „Cryptocurrency value formation: An empirical study leading to a cost of production model for valuing bitcoin". In: *Telematics and Informatics* 34.7 (Nov. 2017), S. 1308–1321.

[55] James Heather und David Lundin. „The Append-Only Web Bulletin Board". In: *Formal Aspects in Security and Trust (Proceedings)*. FAST 2008. (Malaga, Spanien, 9. Okt. 2008). Hrsg. von Pierpaolo Degano, Joshua Guttman und Fabio Martinelli. Lecture Notes in Computer Science 5491. Springer Berlin Heidelberg, 5. Apr. 2009, S. 242–256. ISBN: 978-3-642-01465-9.

[56] Ethan Heilman, Alison Kendler, Aviv Zohar und Sharon Goldberg. „Eclipse Attacks on Bitcoin's Peer-to-Peer Network". In: *USENIX Security Symposium (Proceedings)*. USENIX SS 2015. (Washington, DC, USA, 12. Aug. 2015). USENIX Association, 2015, S. 129–144. ISBN: 978-1-931971-23-2.

[57] Martin Hirt und Kazue Sako. „Efficient Receipt-free Voting Based on Homomorphic Encryption“. In: *Advances in Cryptology (Proceedings)*. EUROCRYPT 2000. (Bruges, Belgien, 14. Mai 2000). Hrsg. von Bart Preneel. Lecture Notes in Computer Science 1807. Springer Berlin Heidelberg, 2000, S. 539–556.

[58] Martin Holland. *NSA manipuliert per Post versandte US-Netzwerktechnik*. heise online. 13. Mai 2014. URL: https://heise.de/-2187858 (besucht am 27. 04. 2018).

[59] Martin Holland. *Russland: Telegram-Blockade behindert Hunderte Webdienste – nur nicht Telegram*. heise online. 27. Apr. 2018. URL: https://heise.de/-4036708 (besucht am 27. 04. 2018).

[60] Intel Corporation. *Hyperledger: Sawtooth Documentation*. Version 0.8.13. 2018. URL: https://sawtooth.hyperledger.org/docs/core/releases/latest/contents.html (besucht am 19. 01. 2018).

[61] Markus Jakobsson, Ari Juels und Ronald L. Rivest. „Making Mix Nets Robust for Electronic Voting by Randomized Partial Checking“. In: *USENIX Security Symposium (Proceedings)*. USENIX SS 2002. (San Francisco, CA, USA, 5. Aug. 2002). USENIX Association, 2002, S. 339–353.

[62] Ari Juels, Dario Catalano und Markus Jakobsson. „Coercion-Resistant Electronic Elections“. Extended Abstract. In: *ACM Workshop on Privacy in the Electronic Society (Proceedings)*. WPES 2005. (Alexandria, VA, USA, 7. Nov. 2005). ACM, 2005, S. 61–70.

[63] Axel Kannenberg. *Kryptogeld: Bitmain kündigt ersten ASIC-Miner für Ethereum an*. heise online. 3. Apr. 2018. URL: https://heise.de/-4010449 (besucht am 11. 05. 2018).

[64] Aggelos Kiayias und Moti Yung. „Self-tallying Elections and Perfect Ballot Secrecy“. In: *Public Key Cryptography (Proceedings)*. PKC 2002. (Paris, Frankreich, 12. Feb. 2002). Hrsg. von David Naccache und Pascal Paillier. Lecture Notes in Computer Science 2274. Springer Berlin Heidelberg, 2002, S. 141–158.

[65] Aggelos Kiayias, Thomas Zacharias und Bingsheng Zhang. „An Efficient E2E Verifiable E-voting System without Setup Assumptions“. In: *IEEE Security & Privacy* 15.3 (9. Juni 2017), S. 14–23. ISSN: 1540-7993.

[66] Aggelos Kiayias, Thomas Zacharias und Bingsheng Zhang. „End-to-End Verifiable Elections in the Standard Model“. In: *Advances in Cryptology (Proceedings)*. EUROCRYPT 2015. (Sofia, Bulgarien, 26. Apr. 2015). Hrsg. von Elisabeth Oswald und Marc Fischlin. Lecture Notes in Computer Science 9057. Springer Berlin Heidelberg, 2015, S. 468–498. ISBN: 978-3-662-46802-9.

[67] Sunny King und Scott Nadal. *PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake*. 19. Aug. 2012. URL: https://peercoin.net/whitepaper (besucht am 19. 01. 2018).

[68] Roland Krummenacher. *Implementation of a Web Bulletin Board for E-voting Applications*. Seminarpaper, Software and Systems, Hochschule für Technik Rapperswil, Schweiz. Abrufbar unter https://security.hsr.ch/msevote/seminar-papers/FS10_Implementation_of_a_WBB.pdf. 2010.

[69] Ralf Kuesters, Tomasz Truderung und Andreas Vogt. „Clash Attacks on the Verifiability of E-Voting Systems“. In: *2012 IEEE Symposium on Security and Privacy (Proceedings)*. IEEE, Mai 2012.

[70] Annabell Kuldmaa. „On Secure Bulletin Boards for E-Voting“. Masterarbeit. Institute of Computer Science, Universität Tartu, Estland, 2017.

[71] Constanze Kurz, Frank Rieger und Rop Gonggrijp. *Beschreibung und Auswertung der Untersuchungen an NEDAP-Wahlcomputern*. Techn. Ber. Berlin: Chaos Computer Club und "Wij vertrouwen stemcomputers niet", 30. Mai 2007. 55 S.

[72] Ralf Küsters, Johannes Müller, Enrico Scapin und Tomasz Truderung. „sElect: A Lightweight Verifiable Remote Voting System“. In: *IEEE Computer Security Foundations Symposium (Proceedings)*. CSF 2016. (Lisabon, Portugal, 27. Juni 2016). IEEE, 2016, S. 341–354.

[73] Ralf Küsters, Johannes Müller, Enrico Scapin und Tomasz Truderung. *sElect: A Lightweight Verifiable Remote Voting System*. Techn. Ber. `https://eprint.iacr.org/2016/438`. Cryptology ePrint Archive, Report 2016/438, 3. Mai 2016.

[74] Leslie Lamport, Robert Shostak und Marshall Pease. „The Byzantine Generals Problem“. In: *ACM Transactions on Programming Languages and Systems* 4.3 (Juli 1982), S. 382–401.

[75] Lexas Information Network. *Stromverbrauch*. 2018. URL: `https://www.laenderdaten.de/energiewirtschaft/elektrische_energie/stromverbrauch.aspx` (besucht am 25. 04. 2018).

[76] Yi Liu und Qi Wang. *An E-voting Protocol Based on Blockchain*. Cryptology ePrint Archive, Report 2017/1043. `https://eprint.iacr.org/2017/1043`. 23. Okt. 2017.

[77] Peter Mahlmann und Christian Schindelhauer. *Peer-to-Peer-Netzwerke*. 1. Aufl. Springer Berlin Heidelberg, 10. Juli 2007. 293 S. ISBN: 978-3-540-33991-5.

[78] Trent McConaghy. *The DCS Triangle. Decentralized, Consistent, Scalable. Pick any two.* The BigchainDB Blog. 10. Juli 2016. URL: `https://blog.bigchaindb.com/the-dcs-triangle-5ce0e9e0f1dc` (besucht am 18. 04. 2018).

[79] Patrick McCorry, Siamak F. Shahandashti und Feng Hao. „A Smart Contract for Boardroom Voting with Maximum Voter Privacy“. In: *Financial Cryptography and Data Security (Proceedings)*. FC 2017. (Sliema, Malta, 3. Apr. 2017). Hrsg. von Aggelos Kiayias. Lecture Notes in Computer Science 10322. Springer International Publishing, 10. Feb. 2017, S. 357–375.

[80] Ralph C. Merkle. „A Digital Signature Based on a Conventional Encryption Function“. In: *Advances in Cryptology (Proceedings)*. CRYPTO 1987. (Santa Barbara, CA, USA, 16. Aug. 1987). Hrsg. von Carl Pomerance. Lecture Notes in Computer Science 293. Springer Berlin Heidelberg, 1988, S. 369–378.

[81] Satoshi Nakamoto. *Bitcoin: A Peer-to-peer Electronic Cash System*. 1. Nov. 2008. URL: `https://bitcoin.org/bitcoin.pdf` (besucht am 10. 01. 2018).

[82] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller und Steven Goldfeder. *Bitcoin and Cryptocurrency Technologies. A Comprehensive Introduction.* 1. Aufl. Princeton University Press, 11. Juli 2016. 336 S. ISBN: 978-0-691-17169-2.

[83] Kartik Nayak, Srijan Kumar, Andrew Miller und Elaine Shi. „Stubborn Mining: Generalizing Selfish Mining and Combining with an Eclipse Attack“. In: *IEEE European Symposium on Security and Privacy (Proceedings).* EuroS&P 2016. (Saarbrücken, Deutschland, 21. März 2016). https://eprint.iacr.org/2015/796. IEEE, März 2016, S. 305–320. ISBN: 978-1-5090-1751-5.

[84] Binh Nguyen, Chris Ferris, Gabor Hosszu, Gari Singh, Greg Haskins, Jason Yellick, Jim Zhang, Jonathan Levi, Sheehan Anderson, Srinivasan Muralidharan, Tamas Blummer und Yacov Manevich. *Hyperledger: Fabric Documentation.* Version 1.1.0-preview. 2017. URL: https://hyperledger-fabric.readthedocs.io/en/v1.1.0-preview/ (besucht am 19. 01. 2018).

[85] Karl J. O'Dwyer und David Malone. „Bitcoin Mining and its Energy Footprint“. In: *Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (Proceedings).* ISSC 2014/CIICT 2014. (Limerick, Irland, 26. Juni 2013). Hrsg. von Abdulhussain Mahdi und Ciaran MacNamee. Institution of Engineering und Technology, Juni 2014, S. 280–285.

[86] Wakaha Ogata, Kaoru Kurosawa, Kazue Sako und Kazunori Takatani. „Fault tolerant anonymous channel“. In: *Information and Communications Security.* Springer Berlin Heidelberg, 1997, S. 440–444.

[87] Miyako Ohkubo, Fumiaki Miura, Masayuki Abe, Atsushi Fujioka und Tatsuaki Okamoto. „An Improvement on a Practical Secret Voting Scheme“. In: *International Workshop on Information Security (Proceedings).* ISW 1999. (Kuala Lumpur, Malaysia, 6. Nov. 1999). Hrsg. von Masahiro Mambo und Yuliang Zheng. Lecture Notes in Computer Science 1729. Springer Berlin Heidelberg, 1999, S. 225–234.

[88] P4Titan. *Slimcoin: A Peer-to-Peer Crypto-Currency with Proof-of-Burn. Mining without Powerful Hardware.* 17. Mai 2014. URL: https://github.com/slimcoin-project/slimcoin-project.github.io/raw/master/whitepaperSLM.pdf (besucht am 19. 01. 2018).

[89] Pascal Paillier. „Public-Key Cryptosystems Based on Composite Degree Residuosity Classes“. In: *Advances in Cryptology (Proceedings).* EUROCRYPT 1999. (Prag, Tschechische Republik, 2. Mai 1999). Hrsg. von Jacques Stern. Lecture Notes in Computer Science 1592. Springer Berlin Heidelberg, 1999, S. 223–238.

[90] Choonsik Park, Kazutomo Itoh und Kaoru Kurosawa. „Efficient Anonymous Channel and All/Nothing Election Scheme“. In: *Advances in Cryptology (Proceedings).* EUROCRYPT 1993. (Lofthus, Norwegen, 23. Mai 1993). Hrsg. von Tor Helleseth. Lecture Notes in Computer Science 765. Springer Berlin Heidelberg, 1994, S. 248–259.

[91] R. A. Peters. „A Secure Bulletin Board“. Masterarbeit. Department of Mathematics und Computing Science, Technische Universität Eindhoven, 29. Juni 2005.

[92] Michael K. Reiter. „Secure Agreement Protocols: Reliable and Atomic Group Multicast in Rampart“. In: *ACM Conference on Computer and Communications Security (Proceedings)*. CCS 1994. (Fairfax, VA, USA, 2. Nov. 1994). Hrsg. von Dorothy Denning, Raymond Pyle, Ravi Ganesan und Ravi Sandhu. ACM, 2. Nov. 1994, S. 68–80.

[93] Donald Eastlake und Tony Hansen. *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)*. RFC 6234 (Informational). RFC. Mai 2011.

[94] Ronald L. Rivest, Adi Shamir und Leonard M. Adleman. „A method for obtaining digital signatures and public-key cryptosystems“. In: *Communications of the ACM* 21.2 (Feb. 1978), S. 120–126.

[95] Ronald L. Rivest, Adi Shamir und Yael Tauman. „How to Leak a Secret“. In: *Advances in Cryptology (Proceedings)*. ASIACRYPT 2001. (Gold Coast, Australien, 9. Dez. 2001). Hrsg. von Colin Boyd. Lecture Notes in Computer Science 2248. Springer Berlin Heidelberg, 2001, S. 552–565.

[96] Peter Y. A. Ryan, David Bismark, James Heather, Steve Schneider und Zhe Xia. „Prêt à Voter: a Voter-Verifiable Voting System“. In: *IEEE Transactions on Information Forensics and Security* 4.4 (Dez. 2009), S. 662–673.

[97] Bruce Schneier. *Applied Cryptography. Protocols, Algorithms and Source Code in C*. 20th Anniversary Edition. John Wiley & Sons Inc, 20. März 2015. 784 S. ISBN: 978-1-119-09672-6.

[98] Victor Shoup. „Practical Threshold Signatures“. In: *Advances in Cryptology (Proceedings)*. EUROCRYPT 2000. (Bruges, Belgien, 14. Mai 2000). Hrsg. von Bart Preneel. Lecture Notes in Computer Science 1807. Springer Berlin Heidelberg, 2000, S. 207–220.

[99] Atul Singh, Tsuen-Wan Ngan, Peter Druschel und Dan S. Wallach. „Eclipse Attacks on Overlay Networks: Threats and Defenses“. In: *IEEE International Conference on Computer Communications (Proceedings)*. IEEE INFOCOM 2006. (Barcelona, Spanien, 23. Apr. 2006). IEEE, 2006, S. 1–12.

[100] Greg Slepak und Anya Petrova. *The DCS Theorem*. arXiv ePrint Archive. Version 1. 10. Apr. 2017. arXiv: `1801.04335v1 [cs.DC]`.

[101] Yu Takabatake, Daisuke Kotani und Yasuo Okabe. „An Anonymous Distributed Electronic Voting System Using Zerocoin“. In: *IEICE Technical Reports (Proceedings)*. Workshop on Internet Architecture and Applications 2016. (Taipei, Taiwan, 3. Nov. 2016). Institute of Electronics, Information und Communication Engineers (IEICE), 3. Nov. 2016, S. 127–131.

[102] Thomas Tjøstheim, Thea Peacock und Peter Y. A. Ryan. „A Model for System-Based Analysis of Voting Systems“. In: *International Workshop on Security Protocols (Proceedings)*. Security Protocols 2007. (Brno, Tschechische Republik, 18. Apr. 2007). Hrsg. von Bruce Christianson, Bruno Crispo, James A. Malcolm und Michael Roe. Lecture Notes in Computer Science 5964. Springer Berlin Heidelberg, 2010, S. 114–130. ISBN: 978-3-642-17773-6.

[103] Alin Tomescu und Srinivas Devadas. „Catena: Efficient Non-equivocation Via Bitcoin“. In: *IEEE Symposium on Security and Privacy (Proceedings)*. S&P 2017. (San Jose, CA, USA, 22. Mai 2017). IEEE, 26. Juni 2017, S. 393–409. ISBN: 978-1-5090-5533-3.

[104] Scott Wolchok, Eric Wustrow, J. Alex Halderman, Hari K. Prasad, Arun Kankipati, Sai Krishna Sakhamuri, Vasavya Yagati und Rop Gonggrijp. „Security analysis of India's electronic voting machines“. In: *ACM Conference on Computer and Communications Security (Proceedings)*. CCS 2010. (Chicago, IL, USA, 4. Okt. 2010). ACM Press, 2010, S. 1–14.

[105] Scott Wolchok, Eric Wustrow, Dawn Isabel und J. Alex Halderman. „Attacking the Washington, D.C. Internet Voting System“. In: *Financial Cryptography and Data Security (Proceedings)*. FC 2012. (Kralendijk, Bonaire, 27. Feb. 2012). Hrsg. von Angelos D. Keromytis. Lecture Notes in Computer Science 7397. Springer Berlin Heidelberg, 2012, S. 114–128.

[106] Gavin Wood. *Ethereum: A Secure Decentralised Generalised Transaction Ledger. Ethereum Yellow Paper*. Github. Version EIP-150 759dccd. `https://ethereum.github.io/yellowpaper/paper.pdf`. 7. Aug. 2017.

[107] Karl Wüst und Arthur Gervais. *Ethereum Eclipse Attacks*. Techn. Ber. ETH Zurich Department of Computer Science, 2016.

[108] Seda Yanik und Anil Savaş Kiliç. „A Framework for the Performance Evaluation of an Energy Blockchain“. In: *Energy Management - Collective and Computational Intelligence with Theory and Applications*. Hrsg. von Cengiz Kahraman und Gülgün Kayakutlu. Studies in Systems, Decision and Control 149. Springer International Publishing, 2018, S. 521–543.

[109] Filip Zagórski, Richard T. Carback, David Chaum, Jeremy Clark, Aleksander Essex und Poorvi L. Vora. „Remotegrity: Design and Use of an End-to-End Verifiable Remote Voting System“. In: *Applied Cryptography and Network Security (Proceedings)*. ACNS 2013. (Banff, Kanada, 25. Juni 2013). Hrsg. von Michael Jacobson, Michael Locasto, Payman Mohassel und Reihaneh Safavi-Naini. Lecture Notes in Computer Science 7954. Springer Berlin Heidelberg, 2013, S. 441–457.

[110] Zhichao Zhao und T.-H. Hubert Chan. „How to Vote Privately Using Bitcoin“. In: *Information and Communications Security (Proceedings)*. ICICS 2015. (Beijing, China, 9. Dez. 2015). Hrsg. von Sihan Qing, Eiji Okamoto, Kwangjo Kim und Dongmei Liu. Lecture Notes in Computer Science 9543. Springer International Publishing, 5. März 2016, S. 82–96.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift