HOCHSCHULE MITTWEIDA
University of Applied Sciences

MASTERARBEIT

Herr
Martin Schuster, B.Sc.

Analyse, Konzeptionierung, Entwicklung und Evaluation eines dezentralen Feedback-Systems mit einer Blockchain Komponente für die Blockchain Academy Mittweida

Mittweida, September 2022

Fakultät **Angewandte Computer- und Biowissenschaften**

MASTERARBEIT

Analyse, Konzeptionierung, Entwicklung und Evaluation eines dezentralen Feedback-Systems mit einer Blockchain Komponente für die Blockchain Academy Mittweida

Autor:
Martin Schuster

Studiengang:
Medieninformatik und Interaktives Entertainment

Seminargruppe:
MI20w1-M

Erstprüfer:
Prof. Dr.-Ing. Andreas Ittner

Zweitprüfer:
Mario Oettler, Dipl.-Volkswirt

Einreichung:
Mittweida, 30.09.2022

Verteidigung/Bewertung:
Mittweida, 01.11.2022

Faculty of **Applied Computer Sciences and Biosciences**

MASTER THESIS

Analysis, conceptual design, development and evaluation of a decentralized feedback system with a blockchain component for the Blockchain Academy Mittweida

Author:
Martin Schuster

Course of Study:
Applied Computer Science

Seminar Group:
MI20w1-M

First Examiner:
Prof. Dr.-Ing. Andreas Ittner

Second Examiner:
Mario Oettler, Dipl.-Volkswirt

Submission:
Mittweida, 30.09.2022

Defense/Evaluation:
Mittweida, 01.11.2022

Bibliografische Beschreibung:

Schuster, Martin:
Analyse, Konzeptionierung, Entwicklung und Evaluation eines dezentralen Feedback-Systems mit einer Blockchain Komponente für die Blockchain Academy Mittweida. – 2022. – 49 S.
Mittweida, Hochschule Mittweida – University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften, Masterarbeit, 2022.

Referat:

Die vorliegende Arbeit beschäftigt sich mit der Analyse, Konzeption, Implementation und Evaluation eines dezentralen Feedback-Systems für die Blockchain Academy Mittweida. Es wurde nach einer Lösung gesucht, das Feedback für die angebotenen Kurse auf der Seite der Blockchain Academy Mittweida erfassen zu können. Dabei sollte die Anonymität des Nutzers stets gewahrt bleiben, jedoch für den Betreiber die Möglichkeit gegeben sein, unberechtigtes oder doppeltes Feedback erkennen und aussortieren zu können. Diese Anforderungen konnten durch die Linkable-Ring-Signature gewährleistet werden. Bei diesem Verfahren kann der Nutzer stellvertretend für seine Gruppe eine Nachricht signieren. Der Betreiber kann überprüfen, ob eine signierte Nachricht aus dieser Gruppe stammt und ob es bereits eine Nachricht von einem Nutzer eingegangen ist, ohne die Anonymität des Nutzers aufzuheben. Das System wurde möglichst dezentral gestaltet, um keine zentrale Angriffsstelle bieten zu können und sicherheitsrelevante Teile abkapseln zu können. Es wurde ein Smart Contract angelegt, welcher die zum Signieren benötigten öffentlichen Schlüssel der Gruppenmitglieder eines Kurses bereithält. Die zweite Komponente stellt eine Browsererweiterung dar. Mit dieser kann der Nutzer sich in die Feedbackgruppe zu einem Kurs eintragen und seine Schlüsselpaare für die Signatur Generierung speichern lassen. Die dritte Komponente ist ein Plugin auf dem WordPress-System der Blockchain Academy Mittweida, mit welchem das Feedback auf Verknüpfbarkeit geprüft werden kann. Mittels eines funktionalen Testverfahrens und einer Probandengruppe wurde die erarbeitete Lösung untersucht. In beiden Fällen bestand die Lösung die an diese gestellten Anforderungen und wurde positiv von der Probandengruppe aufgenommen. Es wurde sich mehrheitlich für die vorgestellte Lösung und gegen eine Lösung mit einem Kennwort, welches der Nutzer eingeben muss, bevor er eine Umfrage ausfüllen kann, entschieden. Im Wintersemester 2022/2023 an der Hochschule Mittweida könnte die Anwendung zusammen mit dem Masterstudiengang Blockchain & Distributed Ledger Technologies (DLT) in einem größeren Praxistest eingesetzt und weiter erprobt und verbessert werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die Blockchain Academy Mittweida (BCAM), ein Projekt des Blockchain Competenz Centers Mittweida (BCCM) verfolgt das Ziel, Blockchain relevante Inhalte für Lehr- und Lernzwecke bereitzustellen. Es wurde eine Lösung gesucht, welche das Feedback durch die Nutzer zu den einzelnen Kursen aufnehmen kann. Besondere Anforderungen wurden dabei an die Anonymität der Nutzer, die Dezentralisierung der Lösung und das Nutzen einer Blockchain relevanten Technologie gestellt.
Die vorliegende Arbeit soll entsprechend vorstellen, wie solch ein System analysiert, konzipiert, entwickelt und evaluiert wurde. Die Ausgangslage bildet dabei das bestehende WordPress-System der Blockchain Academy Mittweida. In diesem sind unterschiedliche Kurse bereits vorhanden und werden als Teil der Ausbildung im Masterstudiengang Blockchain & Distributed Ledger Technologies (DLT) sowie zur öffentlichen freien Verfügung verwendet.
Das Plugin SurveyJS, welches für die Erstellung und Speicherung der Feedbacks zu den Kursen genutzt wird, erlaubt es nur über das Setzen eines Cookies die mehrfache Abgabe von Feedbacks zu verhindert. Diese können jedoch in jedem Browser ohne großen Aufwand gelöscht werden, was eine erneute Teilnahme ermöglicht. Alternativ sind die Umfragen offen und jeder kann so oft wie möglich seine Stimme abgeben. Dadurch ist die Feedback-Abgabe für den Nutzer zwar anonym, jedoch ist die Sicherheit aufseiten des Betreibers nicht länger gegeben. Da dieser Umstand es erlaubt, dass Nutzer mit einer boshaften Absicht diese Schwäche ausnutzen könnten und das Feedback-System mit irrelevanten und unwahren Ergebnissen füllen könnte, was eine sinnvolle Auswertung erschwert oder unmöglich macht. Eine mögliche Lösung für dieses Problem wäre die Benutzung von einmal gültigen Token beziehungsweise Kennwörtern. Dadurch könnte ein Nutzer genau einmal an einer Umfrage teilnehmen, sofern er sich nicht die Token eines anderen Nutzers zu eigen macht und für den Auswertenden wäre mindestens sichergestellt, dass nur berechtigte Personen ein Feedback abgegeben haben. Der Nutzer kann jedoch bei der Abgabe seines Feedbacks nicht sicher sein, ob dieses nicht zusammen mit seinem Token abgespeichert wird und dadurch Rückschlüsse und eventuelle Konsequenzen auf ihn zurückkommen können.
In einem vorherigen Projekt wurde bereits versucht, diese Problematik zu lösen, indem versucht wurde, das Feedback-System um die Technologie der Ring-Signaturen zu erweitern. Die Ring-Signaturen erlauben es stellvertretend für eine Gruppe von mehreren Personen Daten zu signieren, ohne dabei die eigene Identität preiszugeben. Dabei existieren Varianten, die vollständig anonym sind und welche, die eine Verknüpfung von mehreren Signaturen eines Senders erlauben und dennoch die Anonymität des Nutzers wahren. In dem vorherigen Projekt wurde sich für die Variante der Linkable-Ring-Signature entschieden, da diese die gewünschte Sicherheit gegen die Mehrfachabgabe eines Feedbacks von einem Nutzer beinhaltet. Für die Umsetzung wurde ein REST-API-Server erstellt, welcher die Funktionalität der Signatur, Generierung, Verifizierung und Prüfung der Verknüpfbarkeit übernehmen sollte. Die erarbeitete Lösung konnte technisch die Funktionen umsetzen, es bestand aber die Problematik, dass die Nutzer wichtige Informationen wie etwa den benötigten privaten Schlüssel für die Signierung der Nachricht an den REST-API-Server übergeben hätten müssen, was ein Sicherheitsrisiko darstellt. Aufgrund dessen wurde sich dagegen entschieden, die Lösung im Produktivebetrieb einzusetzen.
Die geplante Lösung soll diesen Ansatz aufgreifen und dezentral umsetzen. Dabei wird auf ein System aus drei verteilten Komponenten gesetzt, welche jeweils nur die von diesen benötigten Daten enthält und bei Bedarf sich die restlichen Daten von den anderen Systemen beschaffen kann. Die erste Komponente stellt die Browser Extension dar. Diese soll die Schlüsselpaare des Nutzers bein-

halten und die Erstellung der Signatur für ein Feedback übernehmen. Die zweite Komponente ist ein Smart Contract auf der Ethereum-Blockchain, welcher die öffentlichen Schlüssel der Teilnehmer eines Kurses beinhaltet. Als dritten Komponente kommt einem Plugin für das WordPress-System zum Einsatz. Mit diesem wird die Speicherung der öffentlichen Schlüssel in den Smart Contract beim Einschreiben in einen Kurs realisiert, sowie das Verifizieren und Überprüfung auf Verknüpfbarkeit der Feedbacks gehandhabt.
Für die Umsetzung gilt es zunächst, sich mit den theoretischen Grundlagen zu beschäftigen, dabei besonders die Funktionsweise und Möglichkeiten von Smart Contracts, damit diese möglichst effizient und kostensparend entwickelt werden können. Des Weiteren ist das Wissen über die Entwicklung von Browser Extensions notwendig, da diese mitunter sehr restriktiv gestaltet sind, um den Nutzer und Entwickler zu schützen. Abschließend gilt es, sich mit der Thematik der Ring-Signaturen auseinanderzusetzen, um ein möglicherweise geeigneteres Verfahren zu finden. Anschließend wird die Anwendung konzipiert. Dabei werden die einzelnen Komponenten genauer beschrieben und die benötigte Funktionsweise sowie Anforderungen an diese festgehalten. Der nächste Schritt besteht darin, die Konzeption umzusetzen. Abschließend wird mittels vordefinierter Testfälle die Anwendung auf ihre Funktionalität geprüft und mit einer Probandengruppe, welcher das System vorgestellt wird, soll die Anwendung evaluiert werden.
Es lässt sich vermuten, dass das System gut bis sehr gut angenommen wird, da das geplante Verhalten der Umsetzung größtenteils im Hintergrund ablaufen wird. Lediglich die Eingabe des Schlüsselpaares in die Browser Erweiterung erfordert eine Aktivität des Nutzers. Da der Nutzer von den restlichen Funktionen der Erweiterung nichts sieht, ist ungewiss, inwieweit das Vertrauen in diese Lösung ausfällt und die damit verbunden Adaptionsrate. Diese Fragen sollen mit der Evaluation versucht werden zu beantworten.

2 Theoretische Grundlagen

In diesem Kapitel werden die, für die vorliegende Arbeit, relevanten Grundlagen der Themengebiete Ring Signaturen, Smart Contracts, Dezentralität und Anonymität im Netz vermittelt. Das Thema der Ring Signaturen wurde bezüglich der Verfahrensweise genauer erläutert, sowie eine Betrachtung der unterschiedlichen Varianten, welche sich entwickelt haben, wurde durchgeführt. Der Abschnitt zu den Smart Contracts soll genauer erläutern, was die Besonderheit dieser ist, wie diese funktionieren und wieso es sinnvoll ist, diese für das Vorhaben zu benutzen. Es wird zudem auf die Dezentralität im Netz eingegangen, warum diese von Vorteil sein kann und was Probleme dabei seien können. Abschließend wird auf die Problematik der Anonymität im Netz eingegangen, welche Vor- und Nachteile diese bietet und weshalb darauf ein besonders Augenmerk gelegt werden soll.

2.1 Ring-Signaturen

Ring-Signaturen wurden 2001 erstmals von Rivest, Shamir und Tauman vorgestellt und ermöglichen es dem Nutzer eine Nachricht stellvertretend für seine zugehörige Gruppe zu signieren. Dabei bleibt die Identität des Unterzeichners stets geheim. Im Gegensatz zu Gruppensignaturen ist dabei keine zentrale Verwaltung nötig, welche die Gruppe erstellt und verwaltet. Der dadurch mögliche Widerruf der Anonymität durch den Gruppenleiter ist so ohne weiteres bei Ring-Signaturen nicht möglich. Die Zuweisung von Mitgliedern zu einem Ring kann ohne das Wissen dieser geschehen.[1]
Die Gruppe an möglichen Unterzeichnern wird als Ring bezeichnet, der signierende wird als Unterzeichner betitelt und alle anderen als Nicht-Unterzeichner. Eine Voraussetzung für dieses Verfahren ist, dass alle Beteiligten über einen öffentlichen Schlüssel P_k, welcher das Signaturverfahren und seinen Verifikationsschlüssel beinhaltet, verfügen. Der zugehörige geheime Schlüssel wird als S_k bezeichnet. Ein Ring-Signaturverfahren wird durch zwei Prozessschritte definiert.[1]

1. Ring-Unterzeichnung (ring-sign(m,P_1,P_2,...,P_r,z,S_z)), diese produziert die Ring-Signatur σ für die Nachricht m, mit den gegebenen öffentlichen Schlüsseln P_1,P_2,...,P_r der r Ring Mitglieder, zusammen mit dem geheimen Schlüssel S_z des Unterzeichners z.
2. Ring-Signatur-Verifizierung (ring-verify(m, σ)) diese nimmt die Nachricht m und die Signatur σ entgegen und gibt ein wahr oder falsch aus, je nachdem ob es eine valide Signatur ist oder nicht.

Der Originalentwurf von Rivest, Shamir und Tauman hatte das Hauptaugenmerk auf der Anonymität des Unterzeichners und sah keinen Widerruf dieser Anonymität vor und wurde im Zusammenhang von sogenannten Whistleblowern vorgestellt. Diese versuchen oft im Sinne der Allgemeinheit geheime Informationen, von Politik und Wirtschaft preiszugeben. Da häufig schwerwiegende Strafen auf solchen Aktionen liegen, ist die Anonymität dieser Whistleblower besonders wichtig. Da diese Eigenschaft nicht für alle Anwendungszwecke zielführend beziehungsweise nützlich ist, wurde das Verfahren erweitert. Mit den Linkable-Ring-Signaturen beispielsweise wurde eine Nachvollziehbarkeit, ob ein Mitglied bereits eine Nachricht aus der Gruppe gesendet hat, integriert.[2] Es haben sich noch weitere Varianten wie Tracable-, Threshold- oder Revocable-Ring-Signaturen entwickelt und zu diesen jeweils noch weitere Abwandlungen, sodass es bereits eine Vielzahl an Varianten gibt, die für unterschiedliche Einsatzzwecke verwendet werden können.

2.1.1 Erläuterung der Funktionsweise

Eine Ring-Signatur besteht aus einer Gruppe von Nutzern, welche über ein öffentliches und geheimes Schlüsselpaar (P_1,S_1),(P_2,S_2),...,(P_n,P_n) verfügen. Für die Signierung einer Nachricht m, benutzt der Unterzeichner z seinen geheimen Schlüssel S_z und die öffentlichen Schlüssel der anderen Nutzer (m, S_z,P_1...P_n). Um eine Ring-Signatur zu erstellen, sind die folgenden fünf Schritte notwendig:

1. Berechne symmetrischen Schlüssel k als Hash der Nachricht m
   k = Hash(m)
2. Unterzeichner wählt zufälligen Initialisierungs-Wert u aus $\{0,1\}^b$, wobei b eine Zahl mit zweier Potenz ist, die größer ist als die verwendeten Schlüssellänge der Nutzer
3. Verschlüsseln von u mit einer symmetrischen Chiffrierungs-Funktion E_k damit v entsteht
   $v = E_k(u)$
4. Für jeden Nutzer außer dem Sender, werden die folgenden Schritte durchgeführt:
   a) Berechne $e = S_i^{P_i} \pmod{N_i}$ wobei S_i eine zufällig generierte Zahl für den privaten Schlüssel und P_i der öffentliche Schlüssel des Nutzers i ist
   b) Berechne nächstes v im Ring mit $v = v \oplus e$ (Exklusiv-Oder-Verknüpfung)
5. Unterzeichner z wird Sz bezeichnet $S_z = (v \oplus u)^d \pmod{N_z}$ wobei d der geheime Schlüssel des Unterzeichners ist

Es wird angenommen die Nutzer Alice, Bob, Eve und Trent bilden eine Gruppe und Bob möchte im Namen der Gruppe eine Nachricht signieren. Dafür wird zunächst ein Zufallswert u bestimmt. Der nächste Schritt besteht darin, für die anderen Mitglieder zufällige geheime Schlüsselwerte s_i zu bestimmen. Der geheime Schlüssel von Bob wird mithilfe einer Falltürfunktion in einen anderen geheimen Schlüssel abgeändert. Anschließend wird ein Hashwert der Nachricht gebildet und anschließend mit einer symmetrische Verschlüsselungsfunktion verschlüsselt. Der daraus entstandene Wert wird mit einer Exklusiven-Oder-Funktion verknüpft. Jeder der Zufallswerte für die anderen Teilnehmer wird dann mit dem öffentlichen Schlüssel des jeweiligen Teilnehmers verschlüsselt. Bob berechnet dann den Wert von y_s, um den Ring zu erstellen (das Ergebnis des Rings muss gleich v sein). Anschließend kehrt er diesen Wert um, um den entsprechenden privaten Schlüssel (x_s) zu erhalten. Bob gibt nun die Gesamtsignatur und die zufälligen x-Werte zusammen mit dem berechneten geheimen Schlüssel frei. Um die Signatur zu überprüfen, berechnet der Empfänger lediglich den Ring und prüft, ob das Ergebnis mit der gesendeten Signatur übereinstimmt. Dieses Verfahren ist teilweise in der Abbildung 2.1 dargestellt. [3]

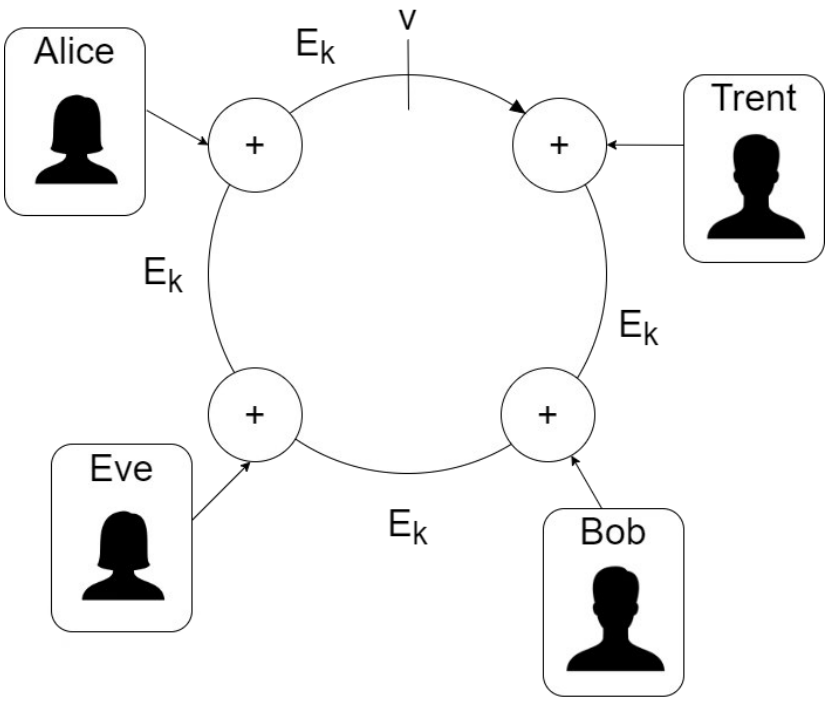


Abbildung 2.1: Ring-Signature

2.2 Ring-Signaturvarianten

Seit dem Entwurf von Rivest, Shamir und Tauman haben sich verschiedene Varianten von Ring-Signatur Verfahren entwickelt. In dem originalen Entwurf stand die Anonymität des Unterzeichners der Nachricht im Vordergrund. Für Anwendungen wie elektronische Wahl- oder Finanzsysteme ist dies jedoch nur ein Faktor. Die Möglichkeit für das doppelte Abgeben einer Wahlstimme in einem elektronischen Wahlsystem wäre katastrophal und würde die Glaubwürdigkeit der Wahl und des Systems nachhaltig stören. Aus diesem Grund haben sich unterschiedliche Ansätze und teilweise mehrere Ableger dieser entwickelt, welche den originalen Vorschlag erweitern. Linkable- [2] und Traceable-Ring-Signaturen [4] ermöglichen das Nachverfolgen, ob ein Nutzer bereits eine Nachricht im Namen seiner Gruppe unterzeichnet hat. Mittels Threshold-Ring-Signaturen wurde die Möglichkeit geschaffen, dass eine festgelegte Mindestanzahl *t* an Mitgliedern einer Gruppe *n* die Nachricht signieren müssen, damit diese als von der Gruppe stammend akzeptiert wird. Nachfolgend werden diese Verfahren genauer erläutert, die Eigenschaften, mögliche Vor- und Nachteile sowie die Einsatzzwecke genauer dargestellt.[5]

2.2.1 Threshold-Ring-Signaturen

Bresson et. al. haben 2002 eine Erweiterung des Entwurfes von Rivest, Shamir und Tauman vorgestellt, bei dem mindestens *t* von *n* Mitgliedern zusammenarbeiten müssen, um eine Signatur zu erstellen. Damit kann gewährleistet werden, dass mindestens *t* Nutzer einer Gruppe *n* zusammengearbeitet haben, damit die Nachricht signiert wurde. Das Ziel ist es, die Glaubwürdigkeit zu steigern. Es bleiben jedoch weiterhin die Identitäten der Unterzeichner und der Subgruppe der *n* Mitglieder geheim.

Unter der Voraussetzung, dass die Beteiligten des Ringes bereits über einen geheimen und öffentlichen Schlüssel verfügen, besteht das Threshold Ring-Signatur Verfahren aus zwei Algorithmen.[5]

1. **T-ring-sign Algorithmus** Als Eingabe wird die Nachricht m, eine Liste (Ring) von n Nutzern mit den jeweiligen öffentlichen Schlüsseln und die geheimen Schlüssel von t Mitgliedern entgegengenommen. Die Ausgabe besteht aus einer (t, a)-Ring-Signatur σ, diese signatur beinhaltet die öffentlichen Schlüssel aller betroffenen Mitglieder.
2. **T-ring-verify Algorithmus** Die Eingabe besteht aus der Nachricht m und der Signatur σ, als Ausgabe wird „wahr“ oder „falsch“ ausgegeben, abhängig davon ob die Signatur zur Nachricht passend ist oder nicht.

2.2.2 Linkable-Ring-Signaturen

2004 wurde von Liu et. al. erstmals das Linkable-Ring-Signature Verfahren vorgestellt, welches die folgenden drei Zielstellungen hatte. Erstens Anonymität beziehungsweise Ununterscheidbarkeit des Unterzeichners, zweitens Verknüpfbarkeit, wenn zwei Signaturen eines Unterzeichners nachvollziehbar zuordenbar sind und drittens Spontanität, welche besagt, dass kein Gruppen-Manager oder Geheimnis nötig ist, um das Verfahren durchführen zu können.[[6]]

Das originale Ring-Signature Verfahren hatte den maximalen Privatsphärenschutz als Grundgedanke. Das ursprünglich erdachte Einsatzgebiet, um Geheimnisse preiszugeben, macht diesen Schutz sehr entscheidend für den Geheimnisenthüller (Whistleblower), da mit dem Preisgeben des Geheimnisses ein hohes Risiko für die Person oder Personengruppen entsteht. In der Theorie ist dies mit Ring-Signaturen möglich, es wird jedoch drauf vertraut, dass Journalisten diesen Informationen blind vertrauen. Mit den Linkable-Ring-Signaturen können von einer Person mehrere Informationen veröffentlicht werden und es kann nachvollzogen werden, dass diese von ein und derselben Person kamen, jedoch bleibt die Anonymität dieser Person weiterhin gewahrt. Dadurch kann die Glaubwürdigkeit der Quelle erhöht werden. Ein weiteres vorgesehenes Anwendungsgebiet ist E-Voting. Es erlaubt dem Wähler anonym eine Stimme abzugeben und kann sicherstellen, dass diese Person nicht noch weitere Stimmen für die gleiche Wahl abgeben kann. Das Einsatzgebiet in der Blockchain Academy Mittweida besteht darin, das Feedback zu Lehrveranstaltungen aufzunehmen. [6]

Durch drei Algorithmen wird das Linkable-Ring-Signatur Verfahren definiert.

1. $(\hat{s},P) \leftarrow G(1^k)$ ist ein probabilistischer Polynomialzeit Algorithmus, welcher einen Sicherheitsparameter k entgegennimmt und den geheimen Schlüssel $\hat{s}$ und öffentlichen Schlüssel P ausgibt.
2. $\sigma \leftarrow S(1^k, \hat{s}, L, m)$ ist ein probabilistischer Polynomialzeit Algorithmus, welcher als Eingabe einen Sicherheitsparameter k, den geheimen Schlüssel $\hat{s}$, eine Liste L von n öffentlichen Schlüsseln, welche den öffentlichen Schlüssel zu $\hat{s}$ enthalten und die Nachricht m enthält, als Ausgabe wird die Signatur σ produziert
3. $1/0 \leftarrow V(1^k, L, m, \sigma)$ ist ein Polynominaler Zeitalgorithmus, welcher als Eingabe Sicherheitsparameter k, eine Liste L von n öffentlichen Schlüsseln, die Nachricht m und die Signatur σ erhält, als Ausgabe wird 1 oder 0 ausgegeben, ob die Signatur zu den Parametern passt

Ein möglicher Nachteil dieses Verfahren ist es, dass die Verknüpfbarkeit nur sichergestellt werden kann, wenn die Liste L der öffentlichen Schlüssel sich nicht ändert beziehungsweise identisch bleibt. Dieser Umstand kann den Einsatz in dynamischen Umgebungen mit wechselnden Mitgliedern des Ringes erschweren.

2.2.3 Tracable Ring-Signaturen

Fujisaki und Suzuki verfolgten einen ähnlichen Ansatz wie Liu et. al. für ein Ring-Signaturverfahren mit eingeschränkter Anonymität. Das Ziel bestand darin, den Missbrauch der uneingeschränkten Anonymität vorzubeugen, in dem mehrmaliges Senden von Nachrichten eines Nutzers erkennbar ist. Dafür wird dem Nutzer ein sogenanntes „Schild“ (engl. Tag) zugeteilt. Dieses beinhalte eine Liste mit den öffentlichen Schlüsseln der Ring-Mitglieder und einer „Ausgabe“ (engl. issue), wofür die Nachricht verwendet wird, beispielsweise eine Abstimmung oder einer Wahl. Dies erlaubt dem Unterzeichner der Nachricht einmal pro „Tag“ anonym die Nachricht zu signieren. Unterzeichnet er eine zweite Nachricht für denselben „Tag“ ist für jeden einsehbar, dass diese zwei Nachrichten zusammengehören, sprich von einem Sender stammen. Dies wurde unter den folgenden vier Sicherheitsanforderungen zusammengefasst.

1. **Öffentliche Nachvollziehbarkeit** Jeder, der zwei Signaturen zu einem Tag erzeugt, kann nachverfolgt werden
2. **„Tag“ Nachvollziehbarkeit** Die maximale Anzahl an Signaturen kann die Anzahl an Ring Mitgliedern nicht übersteigen
3. **Anonymität** Ununterscheidbarkeit der Unterzeichner solange der Unterzeichner nur einmal pro „Tag“ signiert
4. **Exkulpierbarkeit** Ein ehrlicher Unterzeichner kann nicht beschuldigt werden mehrfach für einen „Tag“ signiert zu haben.

Die Tracable-Ring-Signaturen werden durch die folgenden vier Algorithmen definiert.[2]

1. **Gen** - Ein probabilistischer Polynomialzeit Algorithmus, welcher einen Sicherheitsparameter k entgegennimmt und den geheimen Schlüssel $\hat{sk}_i$ und öffentlichen Schlüssel P ausgibt.
2. **Sig** - Ein probabilistischer Polynomialzeit Algorithmus, welcher den geheimen Schlüssel $\hat{sk}_i$, „tag“ $L = (issue, pk_N)$ und Nachricht m als Eingabe entgegen nimmt und die Signatur σ ausgibt.
3. **Ver** - Ein polynominaler Zeitalgorithmus, welcher „tag“ $L = (issue, pk_N)$, Nachricht m und Signatur σ entgegen nimmt und 0/1 ausgibt, abhängig davon, ob die Signatur gültig ist oder nicht.
4. **Trace** - Ein polynominaler Zeitalgorithmus, welcher „tag“ $L = (issue, pk_N)$ und zwei Nachrichten-Signature-Paare $(m,\sigma),(m',\sigma')$ entgegennimmt, als Ausgabe wird „indep“ (unabhängig), „linked“ (verknüpft) oder pk ausgegeben.

2.3 Blockchain und Smart Contracts

In 2008, stellte Satoshi Nakamoto erstmals Bitcoin, die erste Kryptowährung, mit einer Blockchain als verteilte Infrastruktur, vor. Damit wurde es Nutzern erlaubt, ohne einen zentralen Intermediär, wie eine Bank, sicher die als Bitcoin getaufte Kryptowährung zu versenden.[7] 2015 wurde eine weitere Kryptowährung namens Ethereum von Vitalik Buterin veröffentlicht. Diese setzte ebenfalls auf eine Blockchain als Infrastruktur, bot jedoch die Möglichkeit, Smart Contracts einzusetzen. Smart Contracts beinhalten die Vereinbarungen zwischen zwei oder mehr Teilnehmern, werden jedoch automatisch ausgeführt, sobald die vorher festgelegten Bedingungen erfüllt sind. Dadurch können Verträge ohne eine zentrale, vertrauenswürdige Entität geschlossen werden. Um eine Manipulation von Smart Contracts verhindern zu können, sind die Informationen auf jedem Knotenpunkt in der

Blockchain vorhanden und dadurch, dass die Ausführung nicht durch Menschen, sondern an klare Bedingung geknüpfte Logik durch Maschinen ausgeführt wird, kann der menschliche Fehler reduziert werden.[8]
Das Smart Contracts nicht fehlerfrei sind, zeigte eine Manipulation des Smart Contracts der Decentralized Autonomous Organization (DAO) in 2016, wobei durch eine reentrency Attacke 2 Millionen Ether (etwa 50 Millionen US-Dollar) Währung auf der Ethereum Blockchain, gestohlen werden konnten. Es wurde ein Fehler im erstellten Smart Contract ausgenutzt, welcher es erlaubte, beliebig oft eine Transaktion auszuführen, bevor das eigene Konto belastet wurde.[9] Solche meist menschlichen Fehler und die Eigenschaft, dass alles, was in einem Smart Contract geschrieben und gespeichert wird, öffentlich ist, sind die größten Herausforderungen für diese Technologie.

2.3.1 Blockchain

Eine Blockchain kann als verteilte oder duplizierte Datenbank über mehrere Computer (Nodes) hinweg angesehen werden. Das Innovative an dieser Technik ist, dass die Reihenfolge der Transaktionen genau nachverfolgt werden kann, selbst wenn die Nodes sehr weit auseinanderliegen und durch die daraus resultierenden Latenzen sich Unterschiede ergeben, wann welche Transaktion empfangen wurde. Um diese Fähigkeit zu ermöglichen, bestehen die Blockchains in der Regel aus drei Technologien.

- Peer-to-peer Networking: Eine Gruppe von Computern kann ohne eine Zentrale Einheit Kommunizieren.
- Asymmetrische Kryptografie: Beschreibt eine Möglichkeit, eine Nachricht Signiert zu versenden, sodass jeder die Rechtmäßigkeit des Senders überprüfen kann, aber nur der gewollte Empfänger die Nachricht lesen kann. Die meisten Kryptowährungen wie Bitcoin und Ethereum nutzen diese Möglichkeit, damit die Konten der Nutzer sicher sind und nur der Nutzer Token transferieren kann.
- Kryptografisches Hashen: Wird die Möglichkeit genannt, eine einzigartige Identifikationsmöglichkeit für Daten anlegen zu können. Damit größere Datenbestände schnell und einfach abgeglichen werden können und sichergestellt werden kann, dass Daten nicht verändert wurden. Häufig werden dafür sogenannte Merkle Trees verwendet, Transaktionen werden gehashed und ebenenweise bis zur Merkle Root zusammengefasst und wiederum gehashed, eine Veränderung in einem Element würde sich dadurch bemerkbar machen, dass die Merkle Root nicht mehr den richtigen Wert hätte.

Mit diesen drei Elementen lässt sich eine dezentralisierte Datenbank darstellen, welche auf den einzelnen Nodes des Netzwerkes gespeichert wird und die Daten speichert.[10, S.22f]

2.3.2 Asymmetrische Kryptografie

Asymmetrische Kryptografie beschreibt ein System, welches Paare von Schlüsseln benutzt. Diese Paare bestehen aus einem öffentlichen Schlüssel, der bekannt gegeben werden darf und einem privaten Schlüssel, welcher nur dem Besitzer bekannt sein sollte. Diese Schlüssel werden mithilfe von kryptografischen Algorithmen, welche auf mathematischen Ein-Wege-Funktionen basieren erstellt. Die Funktionen sind einfach zu berechnen, lassen sich jedoch nur schwer wieder umkehren. In solch einem System kann jeder, der über den öffentlichen Schlüssel verfügt, Nachrichten für den Empfänger signieren. Jedoch lassen sich diese Nachrichten nur mit dem privaten Schlüssel des

Empfängers wieder entschlüsseln.[11]
RSA (Rivest–Shamir–Adleman) ist eines der bekanntesten und weit verbreitetsten asymmetrischen kryptografischen Verfahren. Die Erzeugung des öffentlichen und privaten Schlüssel findet dabei folgendermaßen statt.[12]

1. Auswahl zweier stochastisch unabhängigen Primzahlen p und q sodass gilt $p \neq q$
2. Berechnung des RSA-Moduls $N = p \cdot q$
3. Berechnung der Eulersche ϕ-Funktion von N $\phi(N) = (p-1) \cdot (q-1)$
4. Wahl einer zu $\phi(N)$ teilerfremden Zahl e, für welche gilt $1 < e < \phi(N)$
5. Berechnung Entschlüsselungsexponenten d mit $e \cdot d \equiv 1 (mod \phi(N))$

2.3.3 Solidity Programmiersprache für Smart Contracts

Solidity ist eine höhere Programmiersprache, welche Ähnlichkeiten zu JavaScript und den C-Sprachen aufweist. Es ist möglich, damit Smart Contracts zu entwickeln und zu EVM (Ethereum Virtual Machine) Bytecode zu kompilieren. Aktuell ist es die meist benutzte Sprache, es gab jedoch auch andere Vertreter wie Serpent (ähnlich zu Python), Lisp-Like Language (LLL) und Mutan, welches bereits eingestellt wurde. Ethereum und Solidity sind beides Open Source Technologien.[10, S.71f]
Entgegen normaler Webanwendungen, bei denen die Bereitstellung und die Skalierbarkeit häufig eine Schwierigkeit ist, ist in Ethereum dies kein großer zusätzlicher Aufwand. Diese verteilten Anwendungen werden auch DApp (decentraliced Application) genannt. Es genügt, die Daten an die EVM (Ethereum Virtual Machine) zu senden und das Programm ist für jeden Nutzer mit einer Ethereum-Wallet auf der Welt verfügbar.[10, S.73f]

2.3.4 Aufbau und Funktionsweise von Smart Contracts

Smart Contracts sind meistens programmiertechnisch Klassen, welche Variablen, Funktionen, Funktionsmodifikatoren, Events und Strukturen enthalten. Eine Funktion, welche die meisten Smart Contracts benutzen, ist die Konstruktor Funktion. Diese wird beim Bereitstellen des Smart Contract auf der Blockchain mittels einer Transaktion ausgeführt und weist den Smart Contract einen Besitzer. Variablen und Funktionen werden genutzt, um Informationen bereitzustellen, beziehungsweise speichern und verändern zu können. Die Variablen lassen sich in Konstante und veränderbare Werte unterscheiden, die Konstanten werden dabei einmalig definiert, wobei die veränderbaren Werte sich ändern können und entsprechend auf der Blockchain geschrieben werden. Funktionen werden ebenfalls in zwei Arten eingeteilt, in schreibende und lesende Funktionen. Lesende Funktionen fragen nur Werte ab von der Blockchain, wohingegen schreibende neue Werte auf eben diese schreiben können. Dafür ist jedoch *gas*, eine Einheit in der der Rechenaufwand gemessen wird eine Aktion durchzuführen, vonnöten. Da jede Transaktion in einem neuen Block erfasst werden muss. Mittels Events lassen sich bestimmte Aktionen ausführen, wenn eine Bedingung erfüllt ist. Diese Events können im Transaktionslog des Blockes, auf dem der Smart Contract liegt, gespeichert werden. Dies hat den Vorteil, dass so Informationen nicht direkt in dem Smart Contract gespeichert werden, was die Kosten intensivste *gas* Operation ist. Dadurch, dass eine Änderung des Smart Contracts nach dem Veröffentlichen auf der Blockchain ohne weiteres nicht möglich ist, muss bereits bei der Erstellung besonders darauf geachtet werden, Fehler zu vermeiden und möglichst für alle Fälle, die eintreten könnten, eine entsprechende Fehlerbehebung eingebaut sein.[13, S.1f]

2.3.5 Factory- und Clone-Factory-Pattern im Kontext von Smart Contracts

Das Konzept hinter dem Factory-Pattern ist, einen Smart Contract als sogenannte Factory anzulegen, welche dann nach Belieben einen vorgegeben Contract veröffentlichen kann, so oft wie dieser benötigt wird. Die Factory kann dabei die Verwaltung dieser erzeugten Instanzen einfacher machen, da alle Adressen in der Factory hinterlegt sind. Da zunächst nur die Factory veröffentlicht werden muss, ist es flexibler und gegebenenfalls besteht die Möglichkeit, *gas* Kosten zu sparen, da nur die Anzahl an Smart Contracts veröffentlicht werden muss, die auch benötigt werden. Ein Nachteil dieses Verfahren ist es, dass die Bereitstellungskosten je Smart Contract immer gleich hoch sind plus die zusätzlichen Kosten für den Factory Contract.[14]
Damit nicht jedes Mal die gesamte Logik neu veröffentlicht werden muss, haben Peter Murray, Nate Welch und Joe Messerman mit dem EIP-1167 „Minimal Proxy Contract“ (*Ethereum Improvement Proposals*) eine Lösung vorgestellt, die dieses Problem beheben soll. Es wird ein Master Contract veröffentlicht, für welchen die Clone beziehungsweise Proxy Contracts erstellt werden. Anfragen und Funktionsaufrufe an diese Proxy Contracts werden an den Master Contract weitergeleitet und dieser leitet die Ergebnisse an die Proxys weiter und diese geben die Ergebnisse dem Aufrufer zurück. Dieses Verfahren wird auch „delegate call“ genannt, Abbildung 2.2 verdeutlicht diesen Umstand. Die Factory produziert schlussendlich die Proxys zu dem Master Contract, was wesentlich weniger *gas* kostet, als jedes Mal einen vollwertigen Smart Contract zu veröffentlichen.[15]

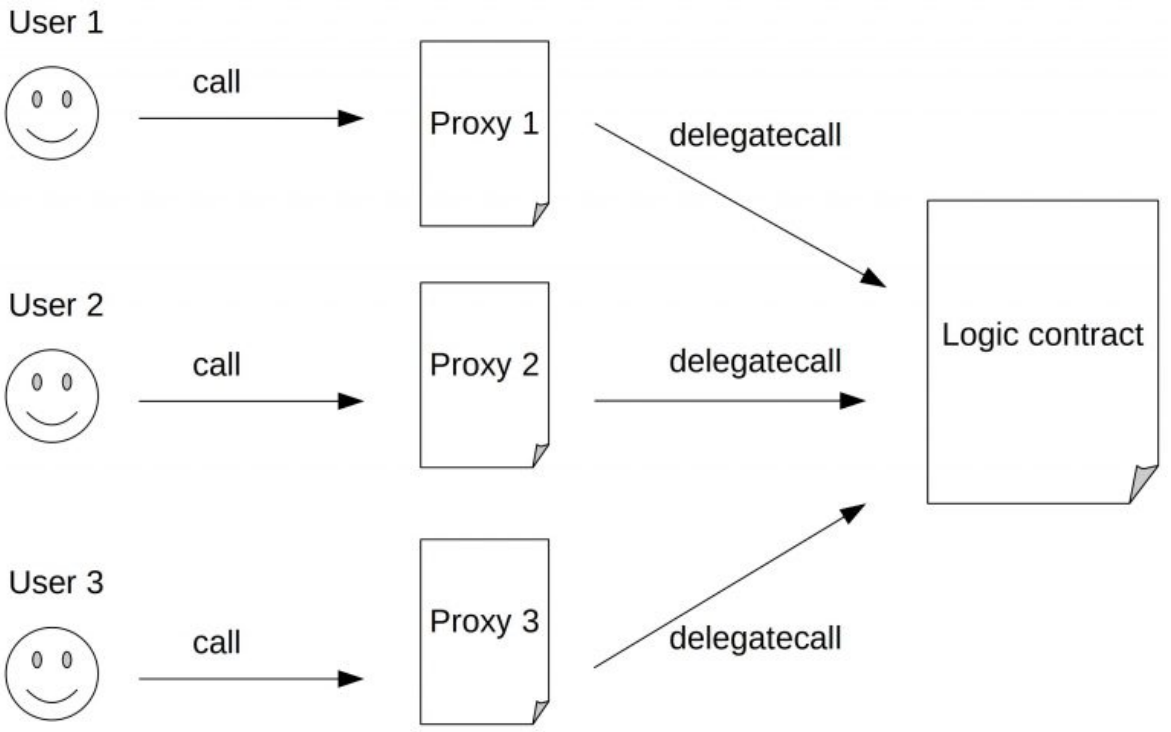


Abbildung 2.2: Verwendung minimal Proxy [16]

2.3.6 GAS Kosten

Das Ausführen von Funktionen von Smart Contracts verbraucht eine bestimmte Menge von *gas*. Die genaue Menge hängt davon ab, wie viel Rechenleistung die Durchführung der Funktion benötigt. In der Abbildung 2.3 ist eine Übersicht gegeben, wie viel die unterschiedlichen Operationen an *gas* als

Kosten verursachen, diese stammt aus dem Yellow Paper zu Ethereum von Gavin Wood. Ethereum folgt einer freien Marktpolitik, wobei die Transaktionsgebühren für eine Transaktion frei festgelegt werden können. Es wird also von der Transaktion in Auftraggeber selbst festgelegt, wie viel er bereit ist, für eine Einheit *gas* zu bezahlen. Transaktionen werden von Sogenannten *Miners* verarbeitet, die Belohnung für diese Verarbeitung kommt aus den Transaktionsgebühren. Was zur Folge hat, dass Transaktionen mit hohen Transaktionsgebühren von den *Minern* bevorzugt und schneller verarbeitet werden. Für die Entwickler von Smart Contracts, ist dieser Umstand eine Herausforderung, da die *gas* Verbrauch erst nach der Verarbeitung der Transaktion bekannt ist und die *Miner* Transaktionen bearbeiten können, wie sie es für richtig erachten. ABDULLAH A. ZARIR et. al. haben in ihrem Paper zum Thema „Developing Cost-Effective Blockchain-Powered Applications: A Case Study of the Gas Usage of Smart Contract Transactions in the Ethereum Blockchain Platform“ diese Problematik untersucht und herausgefunden, dass die meisten *Miner* sich überwiegend an den *gas* Preis der Transaktionen orientieren.[17, S.1]

APPENDIX G. FEE SCHEDULE

The fee schedule G is a tuple of scalar values corresponding to the relative costs, in gas, of a number of abstract operations that a transaction may effect.

Name	Value	Description
G_{zero}	0	Nothing paid for operations of the set W_{zero}.
G_{jumpdest}	1	Amount of gas to pay for a JUMPDEST operation.
G_{base}	2	Amount of gas to pay for operations of the set W_{base}.
G_{verylow}	3	Amount of gas to pay for operations of the set W_{verylow}.
G_{low}	5	Amount of gas to pay for operations of the set W_{low}.
G_{mid}	8	Amount of gas to pay for operations of the set W_{mid}.
G_{high}	10	Amount of gas to pay for operations of the set W_{high}.
$G_{\text{warmaccess}}$	100	Cost of a warm account or storage access.
$G_{\text{accesslistaddress}}$	2400	Cost of warming up an account with the access list.
$G_{\text{accessliststorage}}$	1900	Cost of warming up a storage with the access list.
$G_{\text{coldaccountaccess}}$	2600	Cost of a cold account access.
$G_{\text{coldsload}}$	2100	Cost of a cold storage access.
G_{sset}	20000	Paid for an SSTORE operation when the storage value is set to non-zero from zero.
G_{sreset}	2900	Paid for an SSTORE operation when the storage value's zeroness remains unchanged or is set to zero.
R_{sclear}	15000	Refund given (added into refund counter) when the storage value is set to zero from non-zero.
$R_{\text{selfdestruct}}$	24000	Refund given (added into refund counter) for self-destructing an account.
$G_{\text{selfdestruct}}$	5000	Amount of gas to pay for a SELFDESTRUCT operation.
G_{create}	32000	Paid for a CREATE operation.
$G_{\text{codedeposit}}$	200	Paid per byte for a CREATE operation to succeed in placing code into state.
$G_{\text{callvalue}}$	9000	Paid for a non-zero value transfer as part of the CALL operation.
$G_{\text{callstipend}}$	2300	A stipend for the called contract subtracted from $G_{\text{callvalue}}$ for a non-zero value transfer.
$G_{\text{newaccount}}$	25000	Paid for a CALL or SELFDESTRUCT operation which creates an account.
G_{exp}	10	Partial payment for an EXP operation.
G_{expbyte}	50	Partial payment when multiplied by the number of bytes in the exponent for the EXP operation.
G_{memory}	3	Paid for every additional word when expanding memory.
G_{txcreate}	32000	Paid by all contract-creating transactions after the *Homestead* transition.
$G_{\text{txdatazero}}$	4	Paid for every zero byte of data or code for a transaction.
$G_{\text{txdatanonzero}}$	16	Paid for every non-zero byte of data or code for a transaction.
$G_{\text{transaction}}$	21000	Paid for every transaction.
G_{log}	375	Partial payment for a LOG operation.
G_{logdata}	8	Paid for each byte in a LOG operation's data.
G_{logtopic}	375	Paid for each topic of a LOG operation.
$G_{\text{keccak256}}$	30	Paid for each KECCAK256 operation.
$G_{\text{keccak256word}}$	6	Paid for each word (rounded up) for input data to a KECCAK256 operation.
G_{copy}	3	Partial payment for *COPY operations, multiplied by words copied, rounded up.
$G_{\text{blockhash}}$	20	Payment for each BLOCKHASH operation.

Abbildung 2.3: Übersicht *Gas* Kosten verschiedener Operationen [18]

2.4 Dezentralisierung (Verteiltes System)

Als verteiltes System beschreibt man eine Sammlung an Computerprogrammen, Rechenressourcen über mehrere getrennte Rechenkonten nutzen, um ein gemeinsames Ziel zu erreichen. Diese Trennung kann sowohl softwareseitig als auch hardwareseitig getätigt werden. Das Ziel ist es, Engpässe und Schwachstellen in dem System zu vermeiden. Diese Systeme sind durch die folgenden Eigenschaften gekennzeichnet.[19]

1. Skalierbarkeit: Speicher- sowie Rechenkapazität lässt sich bei Bedarf erweitern, in dem mehr Ressourcen dem System zur Verfügung gestellt werden.
2. Fehlererkennung und Robustheit: Fehler können mitunter einfacher und schneller erkannt und behoben werden
3. Transparenz: Ein Knoten kann mit anderen Knoten im System kommunizieren und auf dessen Ressourcen zugreifen
4. Simultane Verarbeitung: Mehrere Rechner können an ein und derselben Funktion arbeiten, um diese schneller ausführen zu können.
5. Gemeinsame Ressourcennutzung: Ein verteiltes System kann sowohl Hard-, Software als auch Daten gemeinsam nutzen

Der Unterschied zu einem zentralen System besteht darin, dass die Rechenoperationen von einem Computer ausgeführt werden und die Kommunikation zwischen den Knoten des Systems grundlegend unterschiedlich ist. Der Status eines zentralen Systems ist einem zentralen Knoten gespeichert, auf den die Clients je nach Bedarf zugreifen. Je nach Anzahl der Anfragen kann das zu einer Überlastung führen, der zentrale Knotenpunkt ist damit die Schwachstelle des Systems. Ein verteiltes System lagert seine Ressourcen gleichmäßig über das gesamte Netz aus und bietet dadurch keinen zentralen Angriffspunkt. Die Abbildung 2.4 verdeutlicht diesen Umstand. Ein Nachteil dieser Systeme ist die erhöhte Komplexität, was besonders bei der Wartung zur Herausforderung werden kann. Dies ist besonders der Fall, wenn mehrere Teams an unterschiedlichen Komponenten arbeiten und sichergestellt werden muss, dass jedem genau bekannt ist, wofür welche Komponente benötigt wird. Unter den gängigsten Arten von verteilten Systemen fallen die folgenden.[19]

1. Client-Server: Die Darstellung und die Benutzerschnittstelle wird vom Client übernommen und der Server handhabt die Geschäftslogik
2. Mehrstufige Systeme: Ähnlich wie die Client-Server Architektur jedoch verfügt der Server über eine mehrstufige Architektur, welche zusätzlichen Backend-Server für beispielsweise Datenverarbeitung bereithält, um lange Rechenoperationen asynchron durchführen zu können, damit keine Wartezeiten entstehen.
3. Peer-to-Peer Systeme: Jeder Knoten verfügt über die vollständige Instanz einer Anwendung. Präsentation und Datenverarbeitung können von jedem Knoten übernommen werden, dies sorgt für eine enorme Redundanz des Systems.

Verteilte Systeme sind stark verbreitet, Social-Media-Apps, Video-Streaming-Services oder E-Commerce-Webseiten nutzen zu großen Teilen diese Technik.[19]

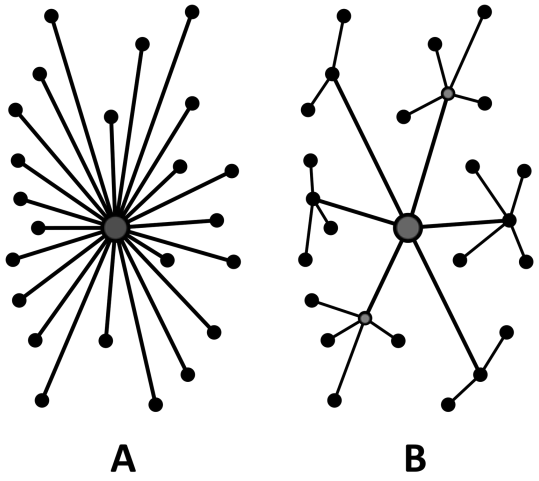


Abbildung 2.4: Darstellung zentralisiertes (A) gegen dezentralisiertes (B) System[20]

2.5 Anonymität im Netz

Mit Anonymität werden häufig sowohl positive als auch negative Gefühle assoziiert. Im Falle von anonymen Wohltätern, anonymen Selbsthilfegruppen oder auch anonymen Warnhinweisen eher positive, wenn es um anonyme Drohungen, anonyme Parteispenden und andere Missetaten geht, eher negative. Dabei ist es häufig keine bewusste Entscheidung, was die Anonymität genutzt wird, sondern eine intuitive, ob jemand namentlich mit einer gewissen Aktion in Verbindung gebracht werden möchte. Mit dieser Entscheidung wird das Datenschutz-Grundrecht auf informationelle Selbstbestimmung ausgeübt. Wäre die Menschen gezwungen, in jeder Lebenslage mit ihren Namen deutlich sichtbar zu erkennen sein, würde dieses Recht verletzt werden und die Person könnte egal bei welcher Gelegenheit nicht sicher sein, wer und was andere Personen über einen wissen. Die anonyme Massengesellschaft bietet dem einzelnen Schutz und Geborgenheit vor unzumutbaren staatlichen und gesellschaftlichen Zugriffen.[21, S.1 ff]
Während im alltäglichen Leben die Anonymität des Einzelnen durchaus möglich ist. Herrscht im Internet häufig das Gegenteil, jede Transaktion eines Nutzers, wird gespeichert an ein oder mehr Stellen, beispielsweise E-Mailverkehr. Selbst das reine besuchen von Webseiten wird von vielen Anbietern gespeichert, verbunden mit Nutzerdaten, um Profile für Nutzer anlegen zu können und zielgerichtete Werbung für diese Schalten zu können. Das Teledienstedatenschutzgesetz räumt jedem das Recht ein, sich anonym durch das Netz zu bewegen. Kein Provider darf aufzeichnen, wer was wann im Netz getan hat, wenn dies nicht ausnahmsweise für Abrechnungszwecke erforderlich ist. Da das Internet jedoch ein globales Netzwerk ist, wird dieses Gesetz von vielen Anbietern missachtet und kann auch nicht durchgesetzt werden. [21, S.6 ff]

3 Konzeption

Mit diesem Kapitel wird, aufbauend auf den theoretischen Grundlagen, ein Konzept entwickelt, wie und in welcher Form das Vorhaben umgesetzt werden kann. Zunächst wird die in einem vorangegangenen Projekt entwickelten Lösung genauer betrachtet, um mögliche Lösungen weiterverwenden zu können oder Ansätze verfolgen zu können, welche sich als vielversprechend herausgestellt hatten. Anschließend wird das geplante dezentrale System theoretisch beschrieben, wie viele Komponenten es haben soll sowie welche Aufgaben und Anforderungen an jede Komponente gestellt werden. Für diesen Zweck werden diese Anforderungen in einem Anforderungskatalog festgehalten, um eine klare Zielstellung bei der Entwicklung haben zu können. In einem abschließenden Schritt gilt es die Evaluation für die Anwendung zu konzipieren, damit geprüft werden kann, inwieweit sich das umgesetzte System bewähren kann.

3.1 Analyse Ausgangslage

In einem vorangegangenen Projekt wurde versucht, ein digitales Meinungsäußerung-System für die Blockchain Academy Mittweida zu entwickeln. Besondere Anforderungen wurden dabei an die Anonymität der Nutzer gestellt sowie eine Integration der Blockchain-Technologie zu schaffen. Für die Anforderung der Anonymität wurde sich für die Technologie der Ring-Signaturen, genauer die Variante der Linkable-Ring-Signaturen entschieden, um die Anonymität der Nutzer zu gewährleisten und um Mehrfachabstimmungen unterbinden zu können. Die Funktionalität wurde als REST-API-Server samt eines dazugehörigen WordPress-Plugins umgesetzt. Um das Verständnis für das Vorhaben zu fördern und die Akzeptanz für diese Lösung zu erproben, wurde ein Erklärvideo angefertigt, welches in einer Evaluation von einer Probandengruppe evaluiert wurde. Dabei fiel das Feedback für das Video wie auch die darin vorgestellte technologische Lösung überwiegend positiv aus, was eine Fortführung der Arbeit rechtfertigte und die Möglichkeit bot, weitere Schritte durchzuführen, um die Dezentralisierung dieses System zu erhöhen. Aufgrund der Komplexität und des knappen Zeitrahmens konnten jedoch nicht alle Bestandteile des Vorhabens vollends umgesetzt werden. Beispielsweise konnte die Nutzung der Blockchain für einen Teil der Aufgaben nicht realisiert werden und ging über die Konzeptionsphase nicht hinaus. Geplant war beispielsweise, dass die Generierung der Signatur durch einen Smart Contract übernommen wird und das signierte Feedback ebenfalls in einem Smart Contract gespeichert werden könnte. Dies würde das Feedback öffentlich, verifizierbar machen und gleichzeitig die Anonymität der Nutzer schützen. Dadurch wäre es ein transparentes-System gewesen, was möglicherweise das Vertrauen in die Lösung gefördert hätte. Das WordPress-Plugin bot nur einen rudimentären Funktionsumfang, in dem es das gespeicherte Feedback mit der Signatur anzeigte. Ein weiterer Punkt war die zu hohe Zentralität der Lösung, welche sich dadurch auszeichnete, dass der Großteil der Funktionalität durch die REST-API erfolgte. Dies bedeutete auch, dass die privaten Schlüssel zum Signieren an den Server gesendet werden mussten, was aus Sicherheitsgründen nicht empfehlenswert ist und so in der Form nicht wiederholt werden sollte.

3.2 Verteilte System

Das Ziel für das geplante Vorhaben ist, dass die Lösung so dezentral wie möglich ist. Dabei soll kein zentrales System benutzt werden, welches den Hauptteil der Funktionen übernimmt und somit ein zentraler Angriffspunkt wäre. Das Vorhaben lässt sich in drei Komponenten aufteilen, welche

an verschiedenen Orten existieren. Die erste Komponente stellt die Browser Extension dar. Diese erlaubt es dem Nutzer, seine privaten und öffentlichen Schlüssel für die Kurse zu verwalten. Dies soll lokal auf dem Computer des Nutzers geschehen. Als zweite Komponente wird ein Smart Contract entwickelt, dieser speichert die öffentlichen Schlüssel der Nutzer pro Kurs und macht diese öffentlich verifizierbar. Der Smart Contract ist auf der Ethereum-Blockchain vorhanden und kann somit von jedem eingesehen und verifiziert werden. Ein Plugin für das WordPress-System stellt die dritte Komponente dar. Diese ermöglicht die Kommunikation mit der Browser Extension und dem Smart Contract. Die Komponenten können für bestimmte Funktionen untereinander kommunizieren und bestimmt Daten austauschen. In der Abbildung 3.1 ist dieses System entsprechend dargestellt. Nachfolgend werden die einzelnen Bestandteile genauer erläutert, welche Funktionalität und Besonderheiten diese aufweisen.

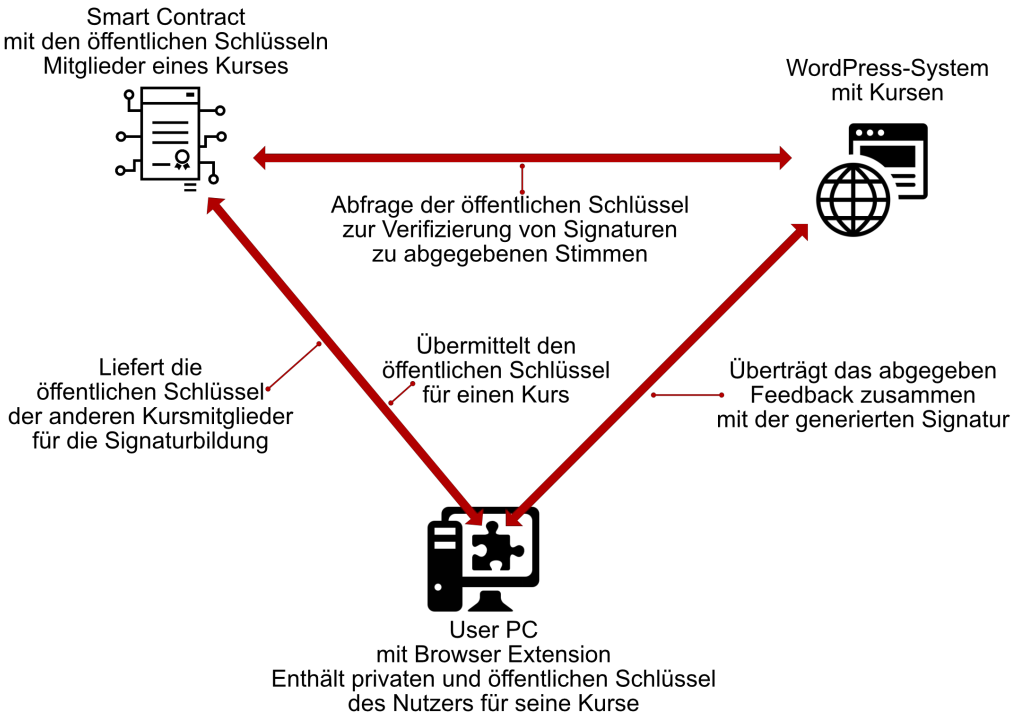


Abbildung 3.1: Konzeptgrafik geplante dezentrale System Architektur

3.2.1 Smart Contract

Der Smart Contract soll die Aufgabe übernehmen, die öffentlichen Schlüssel der Kursteilnehmer bereitzustellen. Dies dient dazu, dass die Mitglieder überprüfen können, ob sie für den jeweiligen Kurs eingeschrieben sind und da die Schlüssel auf einer Blockchain hinterlegt sind, sind diese zudem unveränderbar, was eine mögliche Manipulation an den Nutzern erschweren soll. Die Browser Extension schreibt beim Einschreiben in einen Kurs den öffentlichen Teil des Schlüsselpaares eines Teilnehmers in den Smart Contract. Die Kosten für diese Transaktion sollen vom Betreiber, der Blockchain Academy Mittweida, getragen werden. Das WordPress-Plugin kann die öffentlichen Schlüssel für einen Kurs abfragen, um ein eingereichtes Feedback zu überprüfen.

Für die Umsetzung bietet sich Solidity an, dies ist die Standard-Sprache für die Entwicklung von Smart Contracts auf der Ethereum-Blockchain. Da bereits Vorerfahrungen mit dieser Programmiersprache gemacht wurden und diese der Programmiersprache JavaScript sehr ähnlich ist, ist anzunehmen, dass die Programmierung der Lösung dadurch am effizientesten erfolgen kann. Es bieten sich mehrere Möglichkeiten an, die benötigten Contracts umzusetzen. Die erste Möglichkeit wäre, für jeden Kurs einen eigenen Smart Contract anzulegen, dies hätte den Vorteil, dass die Kurse strikt voneinander getrennt sind, wodurch ein Fehler in einem Smart Contract eines Kurses nicht zwangsläufig auf alle anderen Kurse Auswirkung hat. Als zweite Möglichkeit könnte ein Contract für alle Kurse angelegt werden, dadurch würde nur einmal die Kosten anfallen, um einen Contract zu veröffentlichen. Des Weiteren ist der Einsatz des Factory- und Clone-Patterns unter Umständen sinnvoll.
Die unterschiedlichen Möglichkeiten müssen entsprechend evaluiert werden, um die geeignetste Methode ermitteln zu können. Für diesen Zweck wird ein möglichst realistisches Testszenario entwickelt und die einzelnen Transaktionskosten der Varianten zusammengerechnet und miteinander verglichen, am Ende muss abgeschätzt werden, inwieweit gewisse Vor- und Nachteile von bestimmten Varianten für das geplante Einsatzszenario relevant sind. Aufgrund dieses Ergebnisses wird die zu implementierende Variante gewählt.

3.2.2 Browser Extension

Die Browser Erweiterung stellt die Schnittstelle des Nutzers zum System dar. Mit dieser soll der Nutzer die Kontrolle über zwei wichtige Komponenten behalten. Die erste Komponente besteht daraus, dass der Nutzer sich in eine Feedbackgruppe für einen Kurs einschreiben kann, damit dieser ein anonymes verifizierbares Feedback absenden kann. Dies wird durch die Browser Extension realisiert, damit wäre der private Schlüssel zu jeder Zeit im Besitz des Nutzers und wird nicht extern übermittelt beziehungsweise weiter verwendet. Dies wäre in Bezug zur Sicherheit für den Nutzer bereits eine Verbesserung.
Für die Implementation würde sich für diesen Schritt anbieten, dass mit Aufrufen einer Kursübersichtsseite geprüft wird, ob der Nutzer bereits eingeschrieben ist in die entsprechende Feedbackgruppe für den Kurs. Sollte dies nicht der Fall sein, könnte sich ein Pop-up-Fenster der Browser Erweiterung öffnen, in welcher der Nutzer sein selbst erzeugtes Schlüsselpaar eintragen kann oder eins durch die Anwendung generieren lassen kann. Die Schlüssel würden dabei lokal gespeichert werden. Als Zweites sei der Prozess der Signierung des Feedbacks genannt, bei diesem wird der private Schlüssel benötigt, deshalb findet der Prozess innerhalb der Browser Extension statt und diese fordert die benötigten Informationen bei dem WordPress-Plugin und dem Smart Contract an. Von dem WordPress-Plugin werden die Umfragedaten über eine Schnittstelle abgefragt, sofern das Umfrage-Plugin SurveyJS keine eigene Möglichkeit bietet, diese Daten abzurufen. Die Verbindung zum Smart Contract und das Abrufen der darin gespeicherten öffentlichen Schlüssel der Gruppe wird mittels Web3JS, einer Sammlung von Bibliotheken, welche es erlauben, mit lokalen oder entfernen Ethereum Nodes zu interagieren, umgesetzt.[22]
Die Umsetzung wird zunächst auf den Chrome Browser beziehungsweise Browser, welche die Chrome-Engine nutzen, beschränkt. Dies bedingt sich dadurch, dass Browser mit Chrome-Engine am weitesten verbreitet sind sowie, dass die Nutzer der Blockchain Academy zum größten Teil diese Browser verwenden, 78% der Nutzer verwenden Browser, die auf der Chrome-Engine basieren. In der Abbildung 3.2 ist die Verteilung der genutzten Browser Engines für die Blockchain Academy Mittweida zu sehen für den Zeitraum von September 2021 bis September 2022. Die Entwicklung

von Browser Erweiterungen findet dabei mittels JavaScript, HTML und CSS statt. Es wird sich dabei nach den aktuellen Vorgaben der Manifest V3 Spezifikationen von Google gerichtet, da neue Browser Extension mit älteren Spezifikationen seit dem 17. Januar 2022 nicht mehr für die Veröffentlichung im Chrome-Web-Store zugelassen sind und ab 2024 gar nicht mehr unterstützt werden.[23]

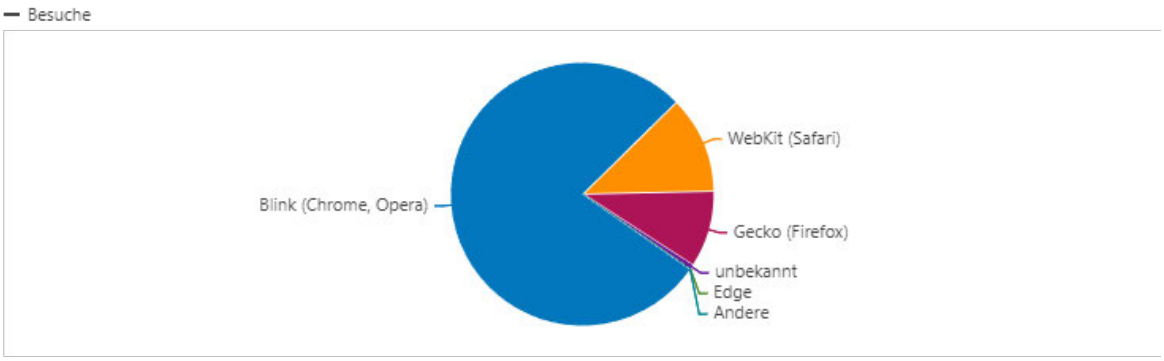


Abbildung 3.2: Browser Nutzung Blockchain Academy Mittweida. (Quelle Matomo Analytics Tool)

3.2.3 WordPress-Plugin

Durch das WordPress-Plugin sollen die entsprechenden Komponenten des WordPress-Systems, das Umfrage Plugin SurveyJS und das Kurs-Plugin LearnDash, mit der Browser Extension und dem Smart Contract kommunizieren können, sofern diese keine geeigneten Möglichkeiten bereitstellen, um benötigte Informationen abfragen zu können.
Die Blockchain Academy Mittweida nutzt WordPress für die Darstellung der Inhalte. Dabei handelt es sich um ein Open-Source Content-Management-System (CMS), welches 2003 von Matthew Mullenweg entwickelt wurde. WordPress ist ein häufig genutztes CMS und 43% der Webseiten im Internet von Hobbyblogs bis professionelle Nachrichtenagenturen nutzen dieses System. WordPress bietet die Möglichkeit, mittels eigener oder erwerbbarer sogenannter Themes das Aussehen an die eigenen Bedürfnisse anzupassen und durch die Möglichkeit, Plugins zu entwickeln beziehungsweise aus einer Bibliothek den Funktionsumfang für die eigenen Anforderungen zu erweitern. Plugins für WordPress werden mittels PHP, HTML, CSS und Javascript realisiert.[24]
SurveyJS ist eine Open-Source JavaScript Bibliothek, welche es erlaubt, Umfragen, Quizze und Formulare zu erstellen und auszuwerten. Es wird direkt vom Entwickler ein Plugin für die Content-Management-Plattform WordPress angeboten, mit welchem ein entsprechendes User-Interface für die Administratoren der Seite aktiviert wird, das zum einfachen Erstellen der Umfragen genutzt werden kann.[25]
Die Kurse, welche auf der Blockchain Academy Mittweida veröffentlicht werden, sind mit dem LearnDash E-Learning Plugin für WordPress erstellt worden. LearnDash bringt modernste E-Learning-Methodik in WordPress ein. LearnDash wird von großen Universitäten, kleinen und mittelständischen Unternehmen, Start-ups, Unternehmern und Bloggern auf der ganzen Welt, für ihre Lernprogramme eingesetzt.[26]
Plugins bieten, häufig bereits Hooks oder Actions an, welche das Ausführen von Funktionen aufgrund bestimmter Aktionen ermöglichen, diese müssen entsprechend für die Plugins erfasst werden beziehungsweise geschaffen werden, damit sich ein Nutzer in eine Feedbackgruppe für einen Kurs eintragen kann und das abgegebene Feedback zu einem Kurs an die Browser Extension gesendet werden kann oder von dieser ausgelesen werden kann, um die Signatur erstellen zu können.

3.3 Anforderungskatalog

Der Anforderungskatalog 3.1 beinhaltet die funktionellen Anforderungen an die einzelnen Komponenten des dezentralen Feedbacksystems. Damit werden die Ziele, welche für die einzelnen Komponenten erreicht werden sollen, formal festgelegt. Ein Anforderungskatalog ist die detaillierte Funktionsbeschreibung, was ein Produkt können muss. Er wird zu Beginn einer Softwareauswahl oder Softwareentwicklung erstellt und fasst präzise zusammen, welche Anforderungen das betreffende Softwaresystem leisten muss.[27] Abweichungen vom Funktionsumfang der Komponenten sowie Änderungen, die während der Entwicklung auftreten, werden in den entsprechenden Kapiteln zur Implementation erläutert.

Tabelle 3.1: Anforderungskatalog

Nummer	Beschreibung
	1. Smart Contract
1.1	Öffentliche Schlüssel müssen eindeutig einem Kurs zugeordnet werden können.
1.2	Das Eintragen eines neuen Schlüssels in den Smart Contract darf nur durch den Besitzer des Smart Contract geschehen.
1.3	Smart Contract soll so effizient wie möglich gestaltet sein.
	2. Browser Extension
2.1	Die Browser Extension soll die Möglichkeit dem Nutzer geben, sich in eine Feedback-Gruppe für einen Kurs einzutragen.
2.2	Die Extension bietet dem Nutzer die Möglichkeit, ein Schlüsselpaar generieren zulassen oder fordert den Nutzer auf, dieses Schlüsselpaar selbstständig einzutragen.
2.3	Die Browser Extension signiert das abgegebene Feedback mittels des Linkable-Ring-Signature Verfahren und sendet die Signatur an das WordPress-System zur Speicherung.
	3. WordPress-Plugin
3.1	Das Plugin bietet die Möglichkeit zu verifizieren, ob ein Feedback von einer berechtigten Person abgegeben wurde.
3.2	Es überprüft, ob von einem Nutzer mehrfach Feedback zu einer Umfrage abgegeben wurde.
3.3	Mittels des Plugins werden die Umfrage-Ergebnisse der Browser Extension bereitgestellt.

3.4 Evaluation

Für die Untersuchung, ob das geplante System von den Nutzern angenommen wird und ob der geplante Funktionsumfang erreicht werden konnte, wird eine zweigeteilte Evaluation durchgeführt.
Der erste Teil besteht darin, anhand von Testfällen die Funktionalität der erarbeiteten Lösung überprüfen zu können. Dabei werden verschiedene Testfälle beschrieben, die im Praxiseinsatz auftreten können sowie die zu erwartende Reaktion des Systems auf diese. Idealerweise finden die Reaktion des Systems in der erwarteten Art und Weise statt, sodass keine weiteren Änderungen vorerst am System zu tätigen sind. Weicht die Reaktion deutlich von dem zu Erwartenden ab, muss entsprechend an den betroffenen Komponenten nachgebessert werden.
Mit dem zweiten Teil der Evaluation soll einer möglichst breit gefächerten Probandengruppe die erarbeitete Lösung kurz vorgestellt werden. Anschließend werden diese gebeten, die Browser Extension

zu installieren und mit dieser ein Feedback für einen Beispielfragebogen auszufüllen. Mit Abschließen der Aufgabe werden die Teilnehmer gebeten, eine Umfrage zu dem Erlebten auszufüllen. Mit diesen Fragen sollen zunächst demografische Daten wie Geschlecht, Alter und Bildungsstand erfasst werden, wodurch überprüft werden soll, ob bei einer demografischen Gruppe die Anwendung besonders gut oder schlecht angenommen wird. Anschließend wird abgefragt, ob es bei der Installation und Verwendung technische Probleme gab und falls ja, welche aufgetreten sind. Abschließend wird noch erfragt, wie den Probanden die Interaktion mit der Anwendung gefallen hat und ob es Verbesserungsvorschläge, Anregungen oder sonstige Unklarheiten gibt. Mit diesen Informationen soll die Anwendung für einen möglichen Praxistest im Wintersemester 2022/2023 an der Hochschule Mittweida im Masterstudiengang Blockchain & Distributed Ledger Technologies (DLT) vorbereitet werden.

3.4.1 Testfälle

In der nachfolgenden Tabelle 3.2 sind die im Vorfeld erwähnten Testfälle, anhand welcher die Anwendung geprüft werden soll, genauer erläutert. Diese sollen Situationen widerspiegeln, welch im produktiven Einsatz der Anwendung eintreten könnten.

3.4.2 Fragebogen

Anhang A zeigt den vorläufigen Fragebogen, mit welchem die Evaluation durchgeführt werden soll. Die Umfrage wird mittels Google Formulare, eine Umfrageverwaltungssoftware, die Teil der kostenlosen, webbasierten Google Docs Editors Suite von Google ist, umgesetzt.[28]

Tabelle 3.2: Testfälle

Nummer	Beschreibung	Zu erwartendes Ergebnis
1	Einschreibung in eine Feedbackgruppe eines Kurses.	Sollte ohne Probleme funktionieren.
2	Mehrmaliges einschreiben in eine Feedbackgruppe.	Sollte ohne die Verwendung eines alternativen Browsers oder Computers nicht möglich sein.
3	Absenden eines Feedbacks ohne installierte Browser Erweiterung.	Feedback sollte wie gehabt aufgenommen und in dem SurveyJS Plugin gespeichert werden.
4	Absenden eines Feedbacks mit installierter Browser Erweiterung, aber ohne in die Feedbackgruppe eingeschrieben zu sein.	Feedback sollte mittels eines Platzhalter-Users signiert werden und das Feedback wird als nicht valide angezeigt.
5	Absenden eines Feedbacks mit installierter Browser Erweiterung und erfolgter Einschreibung in die Feedbackgruppe.	Feedback sollte signiert und als valides Feedback angezeigt werden.
6	Wiederholtes abgeben eines Feedbacks von einem Nutzer zum selben Kurs.	Feedback sollte signiert, als valide angezeigt, jedoch als Duplikat gekennzeichnet werden.
7	Ein weiterer Nutzer schreibt sich in die Feedbackgruppe ein.	Sollte ohne Probleme funktionieren.
8	Nutzer 1 gibt ein Feedback ab.	Feedback sollte signiert werden, ist valide und sollte weiterhin als Duplikat erkannt werden.
9	Nutzer 2 gibt ein Feedback ab.	Feedback sollte signiert werden und ist ein neues Feedback.
10	Nutzer 2 gibt erneut ein Feedback ab.	Feedback sollte signiert werden, ist valide und sollte weiterhin als Duplikat erkannt werden.
11	De- und anschließende Neuinstallation der Erweiterung.	Nutzer können sich wieder in die Feedbackgruppe einschreiben.
12	Nach Neuinstallation und erneutem Einschreiben Abgabe eines Feedbacks.	Feedback sollte signiert werden und ist ein neues Feedback.

4 Implementation

Nachfolgend wird die Entwicklung der drei Komponenten für das dezentrale Feedback-System beschrieben. Die drei Komponenten Smart Contract, Browser Extension und WordPress-Plugin stehen jeweils untereinander in Verbindung und Tauschen entsprechend benötige Informationen aus, sodass jede Komponente nur die benötigten Informationen bereithalten muss und keine Zentralisierung von Informationen entstehen kann. Der Smart Contract stellt die öffentlichen Schlüssel der Teilnehmer eines Kurses bereit. Die Browser Extension erzeugt die Signatur für ein Feedback und behält den privaten Schlüssel des Kursteilnehmers jederzeit im Besitz des Nutzers. Das WordPress-Plugin erlaubt die Verifizierung von eingegangenen Feedbacks der Nutzer.

4.1 Smart Contract

Smart Contracts sind selbst ausführende Verträge, bestimmte Ereignisse lösen vordefinierte Aktionen aus, ohne dass diese explizit überwacht werden. Sobald die Eintrittsbedingungen erfüllt wurden, wird eine entsprechende Transaktion ausgeführt, validiert und anschließend auf der Blockchain geschrieben. Die Aufgabe, die der Smart Contract in dem geplanten System übernehmen sollte, wurde dabei bewusst einfach gehalten. Es sollen zu jedem Kurs die öffentlichen Schlüssel gespeichert werden, welche benötigt werden, um eine Linkable-Ring-Signature für ein eingereichtes Feedback erstellen beziehungsweise verifizieren zu können. Bereits in der Konzeptphase stellte sich heraus, dass dafür sich mehrere Möglichkeiten anboten, wie dieser Smart Contract umgesetzt hätte werden können. Aufgrund der funktionellen Einfachheit des Smart Contracts wurden acht Varianten umgesetzt, welche in der nachfolgenden Tabelle 4.1 genauer beschrieben sind. Die Entwicklung der unterschiedlichen Varianten wurde in Solidity mithilfe der Remix Ethereum IDE umgesetzt. Diese ist eine Online Entwicklungsumgebung, welche die Entwicklung und Veröffentlichung von Smart Contracts auf der Ethereum-Blockchain im Mainnet oder einem der Testnetze erlaubt.[29] Während der Entwicklung wurde das Rinkeby Testnetz verwendet.
Nachfolgend werden die Varianten A, D, E und H genauer erläutert, wie diese umgesetzt wurden und was die Besonderheiten bei diesen sind. Die restlichen Varianten sind nur kleinere Abweichungen der vorgestellten Varianten, wodurch eine genauere Erläuterung sich erübrigt.

Tabelle 4.1: Smart Contract Varianten

Variante	Beschreibung
A	Ein Vertrag pro Kurs, Speicherung der öffentlichen Schlüssel des Nutzers in einem Array im Smart Contract.
B	Ein Vertrag für alle Kurse, der die öffentlichen Schlüssel der Benutzer in einem Mapping mit Kurs-ID und dem dazugehörigen Array öffentlicher Schlüssel speichert.
C	Ein Vertrag pro Kurs, Speicherung der öffentlichen Schlüssel des Benutzers in der Protokolldatei über die Ereignisfunktion.
D	Ein Vertrag für alle Kurse, wobei die öffentlichen Schlüssel der Benutzer und die Kurs-ID im Protokoll über die Ereignisfunktion gespeichert werden.
E	Variante A mit Factory-Pattern.
F	Variante A mit Clone-Factory-Pattern.
G	Variante C mit Factory-Pattern.
H	Variante C mit Clone-Factory-Pattern.

4.1.1 Variante A

Mit der Variante A wird für jeden Kurs ein Smart Contract veröffentlicht und die öffentlichen Schlüssel in einem Array in dem Smart Contract auf der Blockchain gespeichert. Dies hat den Vorteil, dass alle Kurse voneinander getrennt sind, sodass eine Korrumpierung eines Smart Contracts nicht sofort Auswirkung auf alle anderen Verträge hat.
In der Abbildung 4.1 ist der Quellcode des Smart Contracts dargestellt, welcher nachfolgend beschrieben wird. Zeile 1 gibt die Lizenzierung des Smart Contracts an. Die MIT-Lizenz ist eine vom Massachusetts Institute of Technology stammende Open-Source-Lizenz. Sie erlaubt die Wiederverwendung der unter ihr stehenden Software sowohl für Software, deren Quelltext frei verwendbar ist (Open Source) als auch für Software, deren Quelltext nicht frei verwendbar ist (Closed Source). Die Zeile 2 gibt die Version des Compilers des Smart Contracts an, diese legt fest, über welche Funktionen der Smart Contract ohne das Einbinden von externen Bibliotheken verfügen kann. Die Version 0.8.14 war zur Entwicklung die aktuelle stabile Version des Solidity Compilers. Zeile 4 bis 35 beinhaltet den eigentlichen Smart Contract. Zunächst werden drei Variablen definiert, welche für die Verwendung des Smart Contract nötig sind. Die Variable „owner“ vom Typ „address“ speichert die Adresse des Besitzers des Smart Contracts. Für die Speicherung der Kurs ID wird die Variable „courseID“ angelegt, welche vom Typ „uint16“ ist, „uint16“ beschreibt dabei einen Datentyp mit einer Länge von 16 Bit und enthält Zahlenwerte ohne Vorzeichen. Dies ist ausreichend, da die Kurs-ID durch LearnDash bedingt eine vierstellige Zahl ist, welche sich mit 16 Bit darstellen lassen. Mit dem Typ „string[]“ werden die öffentlichen Schlüssel in einem Array unter dem Namen „listPubKeys“ abgespeichert. Die Funktion „constructor“ wird beim Veröffentlichen des Smart Contracts ausgeführt und bekommt als Parameter die entsprechende Kurs-ID, zu dem der Smart Contract zugeordnet wird, übergeben. In dieser Funktion wird die Adresse des Besitzers in die Variable „owner“ geschrieben. Dafür wird die Variable „msg.sender“ genutzt, „msg“ gehört zu den speziellen globalen Variablen, die Eigenschaften enthalten, die den Zugriff auf die Blockchain ermöglichen. Das Attribut „sender“ der „msg“ Variable enthält die Adresse desjenigen, der den Smart Contract veröffentlicht. Abschließend wird die Kurs-ID auf die entsprechende Variable gesetzt. In Zeile 14 bis 17 wird ein benutzerdefinierter „modifier“ „onlyOwner“ definiert. Wird dieser zu einer Funktion hinzugefügt, kann diese nur von dem Besitzer des Smart Contracts ausgeführt werden. Mittels der Funktion „require“ wird geprüft, ob eine Bedingung erfüllt ist, in diesem Fall, ob der Aufrufer der Funktion der Besitzer ist, ist dies der

Fall, wird der Code ausgeführt, andernfalls wird eine Meldung in diesem Fall „Only the owner can add new Public Keys“ ausgegeben und der Vorgang abgebrochen und Änderungen, die bis dahin getätigt wurden, wieder auf den Ursprung geändert. Das Auslagern solch einer mitunter häufig genutzten Bedingung hilft, den Smart Contract so einfach und kurz wie möglich zu halten. Als Nächstes wird mit der Funktion „addPubKey“ die eigentliche Aufgabe des Smart Contracts beschrieben, einen öffentlichen Schlüssel zu einem Kurs eines Teilnehmers dem Contract hinzuzufügen. Diese Funktion bekommt zwei Parameter übergeben, zum einen „courseID_“ vom Typ „uint16“ dieser Parameter wird in Zeile 23 genutzt, um zu überprüfen, ob es sich um den richtigen Smart Contract für den Kurs handelt. Der zweite Parameter namens „pubKey“ vom Typ „string“ hat noch den Zusatz „memory“ dieser gibt an, dass die Variable in einem temporären Speicher zwischengespeichert wird. Dies ist nötig, da der Datentyp „string“ zu groß für den Stack-Speicher ist, in dem die primitiven Datentypen gespeichert werden. Alternativ könnte die Variable auch in dem „storage“ des Smart Contract gespeichert werden, dies ist jedoch aufgrund der dadurch entstehenden Gas-Kosten nicht sinnvoll. Wenn die übergebene Kurs-ID mit der im Smart Contract hinterlegten ID übereinstimmt, wird in das Array der öffentliche Schlüssel gespeichert. Die beiden Funktionen „getPubKeyList“ und „getCourseID“ Zeile 28 und 32 sind dafür zuständig, zum einen das Array der öffentlichen Schlüssel auszugeben und die Kurs-ID anzuzeigen. Mit den drei Modifikatoren „public“ „view“ und „returns“ wird festgelegt, dass diese Funktionen frei ausführbar sind („public“) die Funktion nur von der Blockchain Informationen lesen kann („view“) und es wird der jeweilige Datentyp festgelegt, welchen die Funktion nach einem Aufruf ausgibt („returns“). Damit erfüllt der Smart Contract die von ihm geforderten Funktionen.

4.1.2 Variante D

Im Gegensatz zur Variante A werden in dieser Variante die öffentlichen Schlüssel für alle Kurse in diesem einen Smart Contract gespeichert. Eine weitere Besonderheit ist, dass die Daten nicht in dem Speicher des Smart Contracts gespeichert werden, sondern in der Transaktions-Logdatei. Dies ist deutlich günstiger als das Speichern im Smart Contract beziehungsweise auf der Blockchain. Es wird nachfolgend nur auf die Änderungen am Quellcode eingegangen, da sich einige Komponenten wiederholen beziehungsweise dieselbe Aufgabe haben wie bereits erläutert.
Um ein neues Event einem Smart Contract hinzufügen zu können, muss dieses definiert werden. In der Zeile 11 in Abbildung 4.2 wird dies mit dem Befehl „event“ und dem Bezeichner „NewKey“ getan, dieses Event erhält zwei Parameter „course“ vom Typ „uint16“ und „key“ vom Typ „string“. Diese sind für die Angaben der Kurs-ID und des öffentlichen Schlüssels nötig. Wird dieses Event aufgerufen, schreibt es unter der Bezeichnung „NewKey“ die Kurs-ID und den öffentlichen Schlüssel in die Transaktions-Logdatei. In der Zeile 18 wird die Funktion „logNewKey“ definiert, welche als Parameter die Kurs-ID und den öffentlichen Schlüssel erhält. Mit dem Befehlt „emitt“ und dem Eventbezeichner „NewKey“ wird das Event aufgerufen und ausgeführt. Ein Nachteil dieser Methode ist, dass andere Smart Contracts nicht auf diese Informationen direkt zugreifen können. Für den geplanten Anwendungszweck ist dies jedoch keine Einschränkung, da die Abfrage der Informationen mittel JavaScript und PHP erfolgt und dies ohne Umstände möglich ist.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.14;

contract PubKeyList {
    address owner;
    uint16 courseID;
    string[] listPubKeys;

    constructor(uint16 courseID_) {
        owner = msg.sender;
        courseID = courseID_;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Only the owner can add new Public Keys");
        _;
    }

    function addPubKey(uint16 courseID_, string memory pubKey)
        public
        onlyOwner
    {
        if (courseID_ == courseID) {
            listPubKeys.push(pubKey);
        }
    }

    function getPubKeyList() public view returns (string[] memory) {
        return listPubKeys;
    }

    function getCoureID() public view returns (uint16) {
        return courseID;
    }
}

```

Abbildung 4.1: Quellcode Variante A

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.14;
3
4 contract PubKeyList {
5     address owner;
6
7     constructor() {
8         owner = msg.sender;
9     }
10
11    event NewKey(uint16 course, string key);
12
13    modifier onlyOwner() {
14        require(msg.sender == owner, "Only the owner can add new Public Keys");
15        _;
16    }
17
18    function logNewKey(uint16 courseID_, string memory key) public onlyOwner {
19        emit NewKey(courseID_, key);
20    }
21 }
```

Abbildung 4.2: Quellcode Variante D

4.1.3 Variante E

In Variante A wurde beschrieben, wie der Smart Contract für jeden Kurs gestaltet wurde. Da für jeden Kurs der Smart Contract gleich bleibt, würde sich das in Kapitel 2.3.5 beschriebene Factory-Pattern anbieten. In Variante E wurde dieses genutzt, um eine Schnittstelle zu haben, damit neue Smart Contracts für neue Kurse angelegt werden können. Für die Funktion zum Speichern der öffentlichen Schlüssel wird der gleiche Smart Contract wie in Variante A verwendet, weswegen an dieser Stelle auf die Besonderheiten des Factory Smart Contract eingegangen wird, welcher in Abbildung 4.3 dargestellt ist.
In Zeile 4 wird mittels „import“ der Smart Contract geladen, welcher von der Factory erstellt werden soll. Als Nächstes wird ein Array vom Typ „PubKeyList“, also dem zuvor geladenen Smart Contract angelegt, in welchem später die erstellten Smart Contracts hinterlegt sind. Die Funktion „CreateNewContract“ bekommt als Parameter die Kurs-ID übergeben und erstellt sowie veröffentlicht den neu erstellten Contract und speichert diesen in das zuvor angelegte Array. Mit der Funktion „pklSetter“ lässt sich mithilfe der drei Parameter „_pubKeyListIndex“, „_courseID“ und „_key“ einem entsprechenden Smart Contract an der übergebenen Stelle im Array ein öffentlicher Schlüssel zu einem Kurs zu ordnen. Die „pklGetter“ Funktion fragt die öffentliche Schlüsselliste von einem im Array hinterlegten Smart Contract ab.

```solidity
//SPDX-License-Identifier: Unlicense
pragma solidity ^0.8.14;

import "../PubKeyListSolo.sol";

contract Factory {
    PubKeyList[] public PubKeyListArray;

    function CreateNewContract(uint16 _courseID) public {
      PubKeyList pubKeyList = new PubKeyList(_courseID);
      PubKeyListArray.push(pubKeyList);
    }

    function pklSetter(uint256 _pubKeyListIndex, uint16 _courseID, string memory _key) public {
      PubKeyList(address(PubKeyListArray[_pubKeyListIndex])).addPubKey(_courseID, _key);
    }

    function pklGetter(uint256 _pubKeyListIndex) public view returns (string[] memory) {
     return PubKeyList(address(PubKeyListArray[_pubKeyListIndex])).getPubKeyList();
    }
}
```

Abbildung 4.3: Quellcode Variante E

4.1.4 Variante H

In Variante H wird eine Erweiterung des Factory-Patterns, das Clone-Factory-Pattern genutzt. Um die hohen GAS Kosten zu vermeiden und nicht jedes Mal den Smart Contract mit derselben Logik zu veröffentlichen, wird ein sogenannter „minimal proxy“ bereitgestellt. Dieser leitet lediglich die Anfragen mittels eines „Delegate-Call-Statements “ an den Smart Contract mit der benötigten Logik weiter. Die Änderungen an den Daten bleiben jedoch innerhalb der jeweils genutzten „minimal proxy“ vorhanden. Für den geplanten Einsatzzweck würde für jeden Kurs ein „minimal proxy“ bereitgestellt werden. In Abbildung 4.4 ist der Quellcode für die Clone-Factory dargestellt und wird nachfolgen beschrieben.
Die Variable „baseContract“ beinhaltet die Adresse des Smart Contracts, für welchen die Klone erstellt werden und diese wird mit dem Anlegen des Klone Factory Smart Contracts initial beschrieben. Mittels des Mappings „allclones“ werden die Klone zur jeweiligen Adresse des Aufrufers der Klone-Funktion gespeichert. Das in Zeile 11 definierte Event „NewClone“ wird ausgeführt, wenn ein neuer Klon erzeugt wird. In der Spezifikation EIP-1167 wird empfohlen, dass das Laden des Basis-Contracts in EVM (Ethereum-Virtual-Machine) Code geschieht. Zeile 17 bis 25 beschreibt die Funktion, um einen Klon zu erzeugen. Dabei wird in Assembly Code zunächst der Code des zu klonenden Smart Contracts angegeben, anschließend die Adresse des Contract und als Drittes der Code des Klons geschrieben. Abschließend wird der Klone veröffentlicht und es wird geprüft, ob die Veröffentlichung erfolgreich war. Mit der Funktion „_clone“ wird ein neuer Klon erzeugt und mit der übergebenen Kurs-ID initialisiert. Die Funktion „returnClones“ gibt eine Liste der von einem Besitzer erstellten Klone beziehungsweise „minimal proxy“ zurück.
Der Basis Contract benötigt für das Klonen ein paar Zusätze, diese sind in der Abbildung 4.5 abgebildet. Eine der Änderungen in Zeile 6 ist die Variable „isBase“, wird der Smart Contract alleinstehend bereitgestellt, ist diese Variable auf Wahr („true“) gesetzt und es können keine Daten in den Smart Contract geschrieben werden. Dies ist wichtig, damit beim Aufrufen durch die Klone die Funktionen keine vordefinierten Daten enthalten oder diese im späteren Verlauf gesetzt werden können. Zeile 13 beinhaltet die „initialize“ Funktion, welche aufgerufen wird, wenn der entsprechende Klon erstellt

wird. Es wird zuerst geprüft, ob es sich bei dem Klon um den Basis-Contract handelt (Zeile 15) anschließend wird überprüft, ob der Vertrag bereits initialisiert wurde (Zeile 16), in dem geprüft wird, ob bereits eine Adresse für den Besitzer existiert. Sind beide Bedingungen entsprechend erfüllt, wird der Besitzer und die Kurs-ID mit den übergebenen Werten beschrieben und der Klon kann verwendet werden.

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity >= 0.8.0 < 0.9.0;
3
4  interface PubKeyList {
5      function initialize(address _owner, uint16 _courseID) external;
6  }
7
8  contract CloneFactory {
9      address public baseContract;
10     mapping (address => address []) public allclones;
11     event NewClone (address _newclone, address _owner);
12
13     constructor(address _baseContract) {
14         baseContract = _baseContract;
15     }
16
17     function clone (address _baseContract) internal returns (address instance) {
18         assembly {
19             let ptr := mload (0x40)
20             mstore(ptr, 0x3d602d80600a3d3981f3363d3d373d3d3d363d730000000000000000000000000)
21             mstore(add(ptr, 0x14), shl(0x60, _baseContract))
22             mstore(add(ptr, 0x28), 0x5af43d82803e903d91602b57fd5bf3000000000000000000000000000000000000)
23             instance := create(0, ptr, 0x37)
24         }require(instance != address(0), "ERC1167: create failed");
25     }
26
27     function _clone(uint16 _courseID) external {
28         address identicalChild = clone( baseContract);
29         allclones[msg.sender].push(identicalChild);
30         PubKeyList(identicalChild).initialize(msg.sender, _courseID);
31         emit NewClone( identicalChild, msg.sender);
32     }
33
34     function returnClones (address _owner) external view returns (address [] memory) {
35         return allclones [_owner];
36     }
37 }
```

Abbildung 4.4: Quellcode Clone Factory Variante H

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.14;

contract PubKeyList {
    address public owner;
    bool public isBase;
    uint16 courseID;

    constructor() {
        isBase = true;
    }

    function initialize(address _owner, uint16 courseID_) external {
        // For the base contract, isBase == true. Impossible to use.
        require(isBase == false, "ERROR: This the base contract, cannot initialize"); //Owner address defaults to address(0). Once this function is called, //There is no way to call it again.
        require(owner == address (0), "ERROR: Contract already initialized");
        owner = _owner;
        courseID = courseID_;
    }

    event NewKey(string key);

    modifier onlyOwner() {
        require(msg.sender == owner, "Only the owner can add new Public Keys");
        _;
    }

    function logNewKey(uint16 courseID_, string memory key) public onlyOwner {
        if(courseID_ == courseID){
            emit NewKey(key);
        }
    }
}
```

Abbildung 4.5: Quellcode Variante H

4.1.5 Evaluation der Smart Contract Varianten

Um die verschiedenen Methoden für die Bereitstellung zu vergleichen, die zuvor gezeigt wurden, besonders im Hinblick auf die Kosten, wurden die acht Varianten unseres Smart Contracts jeweils veröffentlichen. Zum Ermitteln der Kosten wurde ein Szenario simuliert, welches fünf Kurse mit je einer Teilnehmerzahl von 20 Bis 40 Stunden pro Kurs vorgesehen hat. Da erste Tests gezeigt haben, dass die Kosten pro Schlüsselspeicherung konstant sind, sofern die Länge des Schlüssels sich nicht ändert, werden 10 Schlüssel über das Rinkeby Testnetzwerk in den Smart Contract geschrieben. Die Kosten für die restlichen Schlüssel sind entsprechend als analog anzusehen. Als öffentliche Schlüssel kommen 512 Bit RSA-Schlüssel im Format x.509 zum Einsatz. Die benötigten Kurs-IDs stellen zufällig generierte 4 Stelle Zahlen dar.
Die Tabelle 4.2 stellt die unterschiedlichen Gesamtkosten der einzelnen Varianten dar. Es ist zu erkennen, dass die Kosten für die Varianten mit der Ereignisfunktionalität (Variante C und D) bei der Speicherung der Informationen erheblich günstiger sind als die Varianten, bei denen die Informationen direkt im Smart Contract gespeichert werden (Variante A und B). Die varianten B und D, bei denen nur einen Smart Contract für alle Kurse verwendet wird, ist ebenfalls günstiger als die entsprechenden varianten A und C. Das liegt daran, dass in dem geplanten Fall die Kosten für die Bereitstellung der Smart Contracts ungefähr gleich bleiben, sodass auf diese Weise keine Kosten eingespart werden können und der Preis für die Speicherung eines neuen Schlüssels bei beiden Varianten gleich hoch ist. Dies führt zu höheren Kosten für variante A und C, da beide Varianten mit der Bereitstellung jedes Kurses Kosten anhäufen. Die Verwendung des normalen Factory-Patterns bietet in dem aufgezeigten Fall keinen Vorteil, da die zusätzlichen Kosten für die Bereitstellung des Factory-Contracts zusammen mit den Kosten für jeden Smart Contract, der über die Factory bereitgestellt wird, etwas teurer ist, da er zusätzlichen Code benötigt, damit er richtig funktioniert. Das Clone-Factory-Pattern ist in diesem Fall etwas interessanter, das Bereitstellen der minimale Proxy ist, wie zu erwarten war, sehr günstig, jedoch bleiben die hohen Anfangskosten durch den Basis Smart Contract und die Schlüsselspeicherung ist insgesamt etwas teurer pro Schlüssel, wodurch die Gesamtkosten ansteigen. Aufgrund dieser Erkenntnisse wurde sich für die Variante D entschieden, für die weitere Entwicklung des Gesamtsystems, da die Kosten am geringsten ausfielen und es vom Funktionsumfang den aktuellen und zukünftigen Anforderungen entspricht. Die als Anhang beiliegende Excel-Datei (Vergleich Smart Contract.xlsx)zeigt die vollständigen Testergebnisse auf.

Tabelle 4.2: Vergleich Gesamtkosten der Smart Contract Varianten

Variante	Gesamtkosten in Eth	Gesamtkosten in Euro
A	0,06040635	61,32 €
B	0,0552133	56,05 €
C	0,0154453	15,68 €
D	0,01216535	12,35 €
E	0,06759345	68,61 €
F	0,06042722	61,34 €
G	0,02114105	21,46 €
H	0,01658626	16,84 €
		Euro - ETH - Preis: 1015,07
		Datum: 30.Juni.2022 - 08:55

4.2 Browser Extension

Die Browser Extension bietet dem Nutzer die Möglichkeit, sich in die Feedbackgruppe der Kurse auf der Seite der Blockchain Academy einzutragen. Für das Abgeben eines anonymen und Duplikation sicheren Feedbacks benötigt der Nutzer jeweils pro Kurs einen privaten und öffentlichen Schlüssel. Die Browser Extension gibt dem Nutzer die Option, beim Eintragen in einen Kurs solch ein Schlüsselpaar zu erstellen, alternativ kann der Nutzer auch ein eigenes Schlüsselpaar in die Eingabemaske eingeben. Dieses wird für ihn in seinem Browser gespeichert. Über ein Popup-Menü werden dem Nutzer die Kurse angezeigt, in welche er eingeschrieben ist und über ein Klick auf solch einen Kurs kommt der Nutzer direkt zu der entsprechenden Kursseite. Die zweite Funktion, welche die Browser Extension übernimmt, ist die der Generierung der Linkable-Ring-Signature. Dafür werden die Daten für die öffentlichen Schlüssel der Gruppe von dem Smart Contract abgefragt und die Feedback-Daten aus dem WordPress-System ausgelesen. Die Generierung der Signatur findet in der Browser Extension statt, da für diese der private Schlüssel des Nutzers benötigt wird, dieser sollte aus Sicherheitsgründen nicht weitergegeben werden.
Für die Entwicklung wurde sich zunächst auf die Browser mit der Chrome-Engine beschränkt, da diese zum überwiegenden Teil (78%) von den Nutzern der Blockchain Academy Mittweida verwendet wird, gefolgt von Safari (Webkit) mit 12% und Firefox mit 10%. Dies ergab die Analyse der Trackingdaten des intern verwendeten Matomo Analytics Tools zum Zeitpunkt der Konzeption. Die Entwicklung erfolgte nach den Spezifikationen der dritten Version, der Manifest V3 vorgaben von Google, da diese den neusten Standard darstellen und Browser Extension mit der zweiten Version zukünftig nicht mehr funktionell sind. Nachfolgend werden die einzelnen Komponenten genauer beschrieben.

4.2.1 Manifest V3

Ein Erweiterungsmanifest gibt dem Browser Informationen über die Erweiterung, beispielsweise die wichtigsten Dateien und die Funktionen, die die Erweiterung verwenden kann. Die Funktionen der Erweiterungsplattform ändern sich, wenn es eine neue Manifestversion gibt. Manifest V3 stellt eine der bedeutendsten Veränderungen in der Erweiterungsplattform dar, seit sie vor einem Jahrzehnt eingeführt wurde. Manifest V3-Erweiterungen genießen Verbesserungen in Bezug auf Sicherheit, Datenschutz und Leistung, sie können auch modernere offene Webtechnologien wie Service Worker und „promises“ verwenden. Seit dem 17. Januar 2022 nimmt der Chrome-Web-Store keine neuen Manifest-V2-Erweiterungen mehr an. Die Erweiterung wurde deshalb in der Version 3 geschrieben. In Abbildung 4.6 sind die einzelnen Bestandteile der Manifest-Datei dargestellt und werden nachfolgend erläutert.
Die ersten 4 Attribute beschreiben die Versionsnummer der Manifest-Datei und damit den Funktionsumfang („manifest_version“), den Namen der Browser Erweiterung („name“), eine Beschreibung wofür die Browser Erweiterung genutzt wird („description“), sowie eine Versionsnummer, in welcher sich die Erweiterung befindet. Mittels des Attributes „icons“, wird das Symbol festgelegt, welches im Browser angezeigt wird, wenn die Erweiterung installiert ist. Es werden dabei drei Größen verwendet, zum einen ein 128 mal 128 große Pixel Variante, welche während der Installation und im Chrome-Web-Store angezeigt wird. Eine 48 mal 48 Pixel Variante, welche auf der Verwaltungsseite der installierten Erweiterungen angezeigt wird und eine 16 mal 16 Pixel große Version, welche das Favicon darstellt, welches für Seiten der Erweiterung genutzt wird. Das „action“ Attribute besitzt zwei unter Attribute, welche zum einen den Titel der Browser Extensiom beim darüber fahren mit der Maus

anzeigen „default_title“, sowie mit „default_popup“ die HTML-Datei die geladen wird, wenn die Browser Extension angeklickt wird. Diese wird in ein Pop-up-Fenster geladen und zeigt dem Nutzer die Kurse an, in denen er eingeschrieben ist. Als Nächstes wird mittels „permissions“ und „host_permissions“ festgelegt, welche Funktionen die Erweiterung ausführen darf, in dem gezeigten Fall sind es das Ausführen von Skripten und der Zugriff auf den Speicher des Chrome-Browsers und auf welchen Seiten die Erweiterung Zugriff hat. Da die Erweiterung nur für die Blockchain Academy Mittweida entwickelt wird, ist an dieser Stelle auch nur diese Domain hinterlegt. Das Attribut „content_scripts“ gibt an, welche JavaScript-Dateien auf einer festgelegten Seite bei Aufrufen dieser geladen werden können. Dabei handelt es sich um fünf Dateien, wobei zwei die konkreten Funktionen zum Einschreiben „enroll.js“ und Erstellen der Signatur zu einem Feedback „survey.js“ beinhalten und drei Dateien, welche zum Erledigen der benötigten Funktion zusätzliche Bibliotheken wie die Web3 Bibliothek für die Verbindung zum Smart Contract und die Funktionalität der Linkable-Ring-Signature, Signierung, Verifizierung und Überprüfung der Verknüpfbarkeit dieser beinhaltet.

```
{
    "manifest_version": 3,
    "name": "Blockchain Academy Mittweida Browser Extension",
    "description": "Browser Extension for Blockchain Academy Mittweida, enables Enrolment into the Feedback System based on Linkable Ring Signatures.",
    "version": "0.1.0",
    "icons": {
        "16": "logo/logo-16.png",
        "48": "logo/logo-48.png",
        "128": "logo/logo-128.png"
    },
    "action": {
        "default_title": "Blockchain Academy Mittweide Explanation",
        "default_popup": "popup/popup.html"
    },
    "permissions": ["scripting", "storage"],
    "host_permissions": [
        "*://localhost:*/*", "https://bccm-s4.hs-mittweida.de/*", "https://blockchain-academy.hs-mittweida.de/*"
    ],
    "background": {
        "service_worker": "service-worker.js"
    },
    "content_scripts": [{
        "js": ["js/enrol.js","js/survey.js", "js/lib/bundle.js", "js/lib/web3.mini.js", "js/lib/jquery.mini.js", "js/lib/md5.min.js"],
        "matches": ["*://localhost:*/*", "https://bccm-s4.hs-mittweida.de/*", "https://blockchain-academy.hs-mittweida.de/*" ]
    }]
}

```

Abbildung 4.6: Quellcode Manifest V3

4.2.2 Einschreiben in das Feedback-System

Die Funktion zum Einschreiben in eine Feedbackgruppe für einen Kurs ermöglicht es dem Nutzer, seinen öffentlichen Schlüssel in diese Gruppe eintragen zu lassen. Dies ist notwendig, damit beim Signieren eines Feedbacks eine Gruppensignatur erzeugt werden kann, welche die Identität des Unterzeichners anonymisiert. Die in der Konzeption geplante Umsetzung, dass sich ein Pop-up-Fenster der Browser Erweiterung öffnet, sobald der Nutzer eine Kursseite öffnet, bei welcher er noch nicht in der Feedbackgruppe eingeschrieben ist, ließ sich aus Sicherheitsgründen nicht realisieren. Das Sicherheitskonzept der Manifest V3 Erweiterungen erlauben nur das Popup-Fenster über das Anklicken des Browsers Extension Icons zu öffnen. Die Lösung, welche aufgrund dieser Einschränkung

erarbeitet wurde, ist in den Abbildungen 4.7 bis 4.10 dargestellt und im Beiliegenden Quellcode (Enrol.js) der Browser Extension einsehbar und wird nachfolgend beschrieben.
Das Einschreiben in einen Kurs wird durch ein zur Laufzeit in die Kursseite injiziertes Einschreibeformular realisiert. Damit dies nicht auf jeder Seite versucht wird zu laden, wird mittels einer Regularexpression geprüft, ob es sich bei einer URL (Uniform Resource Locator) um eine Kursübersichtsseite handelt. Eine Regularexpression ist in der Informatik eine Zeichenkette, die der Beschreibung von Mengen von Zeichenketten mithilfe bestimmter syntaktischer Regeln dient. Ist die aufgerufene Seite eine Kursübersichtsseite, wird als Nächstes die Kurs-ID mittels der in WordPress und LearnDash eingebauten Schnittstelle ermittelt und anschließend geprüft, ob bereits eine Einschreibung in diesen Kurs stattgefunden hat. Dafür werden die lokal gespeicherten Informationen überprüft. Wenn der Nutzer noch nicht eingeschrieben ist, wird wie in Abbildung 4.7 gezeigt eine Schaltfläche angezeigt („Enrol in Feedbackgroup“) um den einschreibe Vorgang zu starten. Wird die Schaltfläche betätigt, es erscheinen zwei Eingabefelder und zwei Schaltflächen. In die Eingabefelder wird jeweils der private beziehungsweise der öffentliche Schlüssel eingetragen, sofern der Nutzer diesen selber bereitstellen möchte, muss dieser in hexadezimaler Form eingetragen werden. Mit der Schaltfläche „Generate Key Pair“ kann der Nutzer sich mit Aktivieren dieser ein entsprechendes Schlüsselpaar erzeugen lassen. Die zweite Schaltfläche „Enrol“ löst die Funktionalität zum Schreiben des Schlüsselpaares in den lokalen Speicher und das Schreiben des öffentlichen Schlüssels in den Smart Contract aus. Zunächst wird überprüft, ob die beiden eingegebenen Schlüssel hexadezimal Werte sind, ist dies nicht der Fall, wird eine entsprechende Meldung an den Nutzer ausgegeben. Anschließend wird ein Objekt erzeugt, welches die Kurs-ID, den Kurstitel, den Link zu dem Kurs und das Schlüsselpaar beinhaltet. Der nachfolgende Schritt besteht darin, über die Schnittstelle von Infura auf den Smart Contract auf dem Rinkeby Testnetz zuzugreifen. Infura bietet Schnittstellen für die Entwicklung und Nutzung der Blockchain an.[30] Um auf den Smart Contract zuzugreifen, muss die Adresse, unter welcher der Smart Contract abrufbar ist und das ABI (Application Binary Interface), diese ermöglicht die Interaktion mit dem Smart Contract und dient zur Kommunikation zwischen zwei Modulen, angegeben werden. Das Schreiben eines neuen Schlüssels in den Smart Contract bedeutet, dass eine Transaktion ausgeführt werden muss, diese kostet eine entsprechende Menge *gas*, eine Gebühr für die Durchführung von Rechenoperationen auf der Blockchain. Infura erlaubt nicht, dass direkt eine Transaktion gesendet werden kann ohne eine Verbindung zu einer Metamask Wallet. Da es jedoch nicht sinnvoll ist, diese Kosten auf die Nutzer zu übertragen, muss die Transaktion manuell erstellt und signiert werden. Dafür wird zunächst für das Einschreiben mit den benötigten Werten ein ABI erzeugt. Anschließend wird die „nonce“ bestimmt, dabei handelt es sich um einen sequenziellen Wert, der angibt, die wievielte Transaktion diese ist. Anschließend wird das Transaktionsobjekt erzeugt. In diesem wird festgelegt, von welcher zu welcher Adresse die Transaktion ausgeführt wird, in diesem Fall ist es ein Testaccount auf dem Rinkeby Netzwerk und die Adresse des Smart Contracts. Zudem wird festgelegt, wie viel GAS die Transaktion maximal kosten darf und es wird der aktuelle GAS Preis ermittelt, zudem werden die vorher erzeugten beziehungsweise ermittelten Daten der nonce und der Transaktionsdaten gesetzt. Anschließend wird die Transaktion mit dem privaten Schlüssel des Testaccounts signiert. Ist die Transaktion erfolgreich, wird das Objekt, welches die Kursinformationen beinhaltet, abgespeichert, der Hash wert, der Transaktion gespeichert und die Seite neu geladen. Der Nutzer ist nun eingeschrieben in die Feedbackgruppe des Kurses und es wird ihm der Hash der Transaktion angezeigt und eine Verlinkung zu Etherscan erstellt, mit der er prüfen kann, ob sein Schlüssel wirklich in dem Smart Contract hinterlegt wurde, dies ist in Abbildung 4.10 zusehen. Ist der Nutzer bereits eingeschrieben, wird, wie in Abbildung 4.10 zu sehen ist, dies in Grün angezeigt(„Enrolled in Feedbackgroup“).

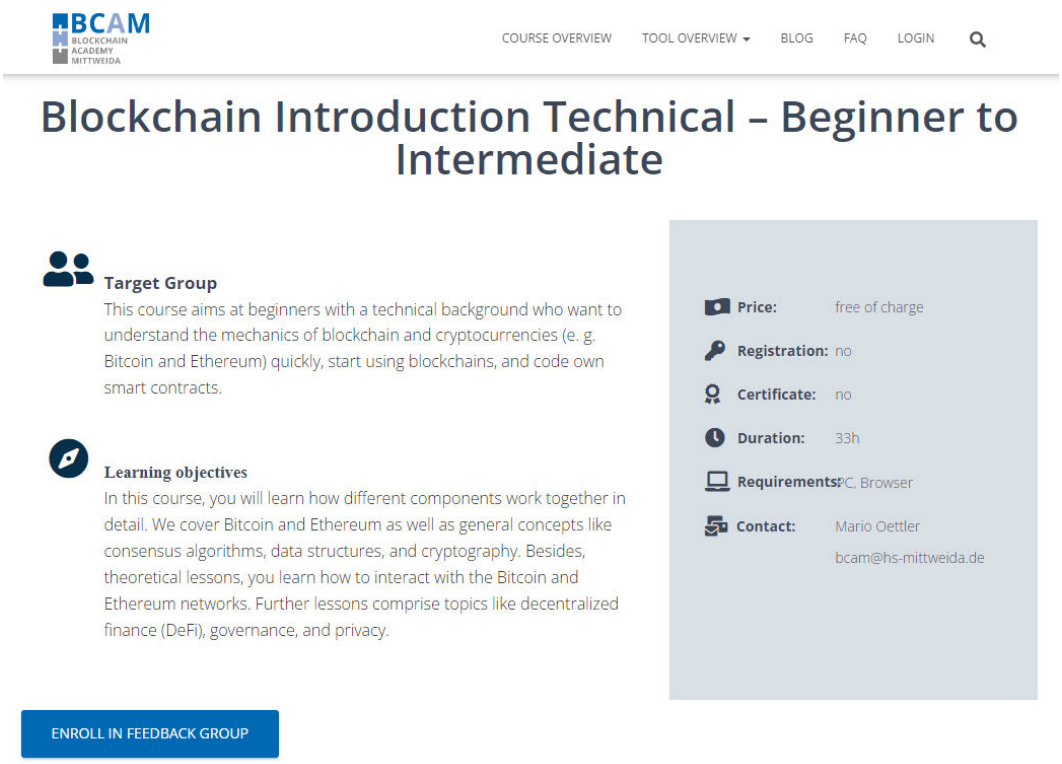


Abbildung 4.7: Einschreibung Schritt 1

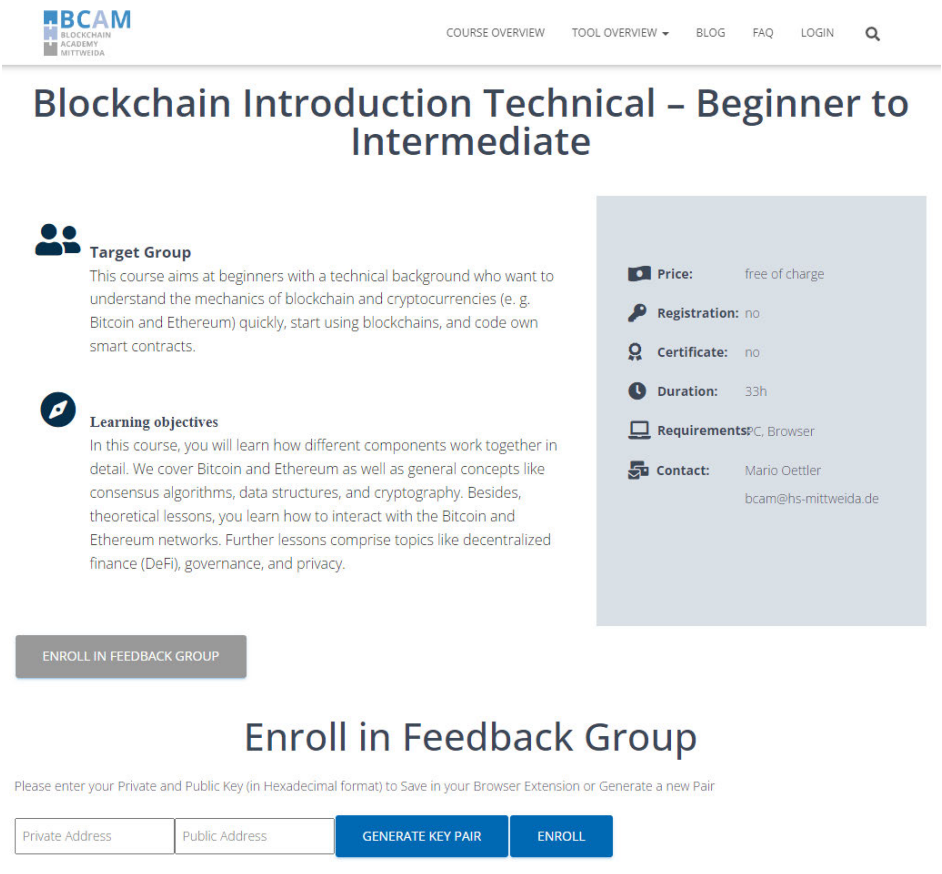


Abbildung 4.8: Einschreibung Schritt 2

Abbildung 4.9: Einschreibung Schritt 3

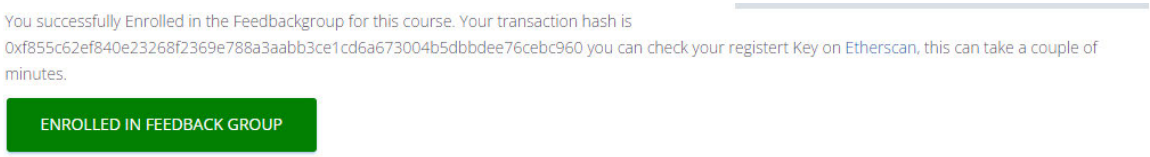


Abbildung 4.10: Einschreibung Schritt 4

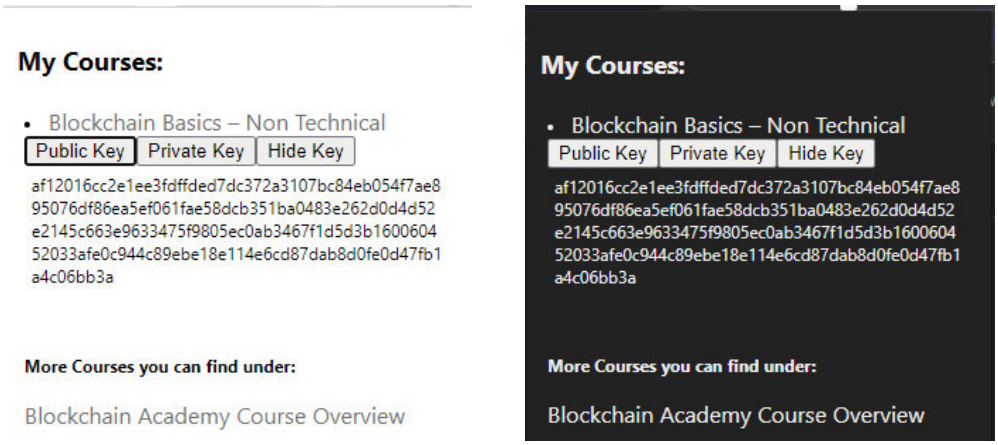
4.2.3 Signatur Generierung

Die Aufgabe der Erstellung einer Signatur zu einem übermittelten Feedback von einem Kursteilnehmer wird durch die Browser Extension realisiert. Es bot sich an, die Funktion an dieser Stelle auszuführen, da dadurch der private Teil des Schlüsselpaares nie den Besitz des Nutzers verlässt. Dies ist wichtig, da sonst Rückschlüsse darauf geschlossen werden könnten, welcher Nutzer welches Feedback übermittelt hat. Der Quellcode für die Funktion befindet sich in der beiliegenden JavaScript-Datei zur Browser Erweiterung „survey.js“ und wird nachfolgend erläutert.
Damit das Feedback signiert werden kann, war es notwendig, die Feedback-Daten an die Browser Extension zu ermitteln beziehungsweise diese abfragen zu können. Um eine Möglichkeit zu haben, zu erkennen, wann ein Feedback gesendet wird, muss eine entsprechende Aktion ausgeführt werden, wenn die Schaltfläche zum Absenden des Feedbacks betätigt wird. Die Browser Extension kann nicht auf die internen Funktionen von WordPress zugreifen, jedoch speichert das Umfrage-Plugin SurveyJS die Umfrage-Daten in einem DIV-Element ab, welches unsichtbar für den Nutzer ist. Ein Div-Element ist einen HTML-Tag, der einen Block oder Bereich kennzeichnet. Dadurch war eine einfache Möglichkeit gegeben, das Feedback auszulesen. Da die entsprechend benötigte Information nicht immer verfügbar ist, sondern nur nach dem Absenden des Feedbacks, muss die Schaltfläche zum Absenden von der Browser Extension abgefragt werden, ob der Nutzer diese betätigt hat. Dies gestaltete sich zunächst schwierig, da SurveyJS das Umfrageformular dynamisch zur Laufzeit lädt und ein entsprechender Zugriff auf das Schaltflächenelement schlugen deswegen fehl, da es noch nicht existierte. Die Lösung für dieses Problem war die Verwendung eines „MutationObservers“ dieser erlaubt es, ein Element des Document Object Model (DOM), welches die Struktur der Webseite als Baumstruktur darstellt, zu beobachten und bei einer Veränderung eine oder mehrere Aktionen auszuführen. In dem gezeigten Fall handelt es sich dabei um das Div-Element mit der ID „main“. Findet der Beobachter die Schaltfläche wird an dieser ein „Eventlistener“ registriert, der bei der Betätigung, die Funktion zur Signierung ausführt. Damit diese Überprüfung nicht auf jeder Seite ausgeführt wird, wird im Vorhinein überprüft, ob es sich bei einer Seite um eine Seite mit einer Umfrage handelt, da alle Seiten mit einer Umfrage jeweils „survey“ als Präfix in der URL der Seite haben, lässt sich darüber die Funktionalität filtern. Betätigt der Nutzer die Schaltfläche zum Absenden eines Feedbacks, liest die Browser Extension die Umfragedaten aus dem unsichtbaren Div-Element aus und ermittelt anschließend die öffentlichen Schlüssel der einzelnen Gruppenmit-

glieder aus dem Smart Contract. Dafür wird die bereitgestellte Schnittstelle durch Infura genutzt, zu dem Smart Contract genutzt. Als Nächstes wird aus den lokal gespeicherten Schlüsselpaaren jenes Schlüsselpaar ermittelt, welches für die Eintragung in den Kurs angelegt wurde. Anschließend wird mittels des Message-Digest Algorithm 5 (MD5) eine weitverbreitete kryptografische Hashfunktion, für die zu signierenden Umfragedaten ein Hashwert erzeugt und anschließend aus dem Gruppenschlüssel, den Daten des Unterzeichners und dem Hashwert der Nachricht die Signatur generiert. Für die Implementation des signier, verifizier und Linkbarkeits-Überprüfung wurde sich für eine Bibliothek von Victor Taelin entschieden.[31]. Diese Bibliothek ermöglicht, die Funktionalität der Linkable-Ring-Signature in JavaScript durchzuführen. Die erstellte Signatur wird anschließend verifiziert, um einer Manipulation, lokal gespeicherten Schlüsseldaten vorzubeugen. Die Signatur und das Ergebnis, ob es sich um ein verifizierbares Feedback handelt, werden anschließend mittels einer AJAX-Funktion in die WordPress-Datenbank des SurveyJS-Plugin geschrieben.

4.2.4 Information-PopUp-Fenster

Mit dem Information-Popup-Fenster werden dem Nutzer die Kurse angezeigt, in die er eingeschrieben ist. Zusätzlich kann er mittels einer Verlinkung direkt zu den einzelnen Kursen gelangen und sich sein Schlüsselpaar für den jeweiligen Kurs anzeigen lassen. Das Popup passt sich dabei an das vom Nutzer präferierte Farbschema an, wie in Abbildung 4.11 zu sehen ist. Der im Anhang beigefügte Quellcode (Popup.js) der Browser Extension enthält die Umsetzung dieser Funktion und wird nachfolgend erläutert.
Das Popup-Fenster besteht aus den drei Dateien „popup.html“, „popup.css“ und „popup.js“. In der HTML Datei wird mittels des HTML5-Templates die Grundstruktur erstellt, damit anschließend die JavaScript-Datei die Informationen an die entsprechenden Stellen schreiben kann. Mit der Cascading-Style-Sheet-Datei wird die Unterscheidung des Hellen- und Dunklenfarbmodus realisiert, die restlichen Angaben passen das Aussehen an. Die JavaScript-Datei „popup.js“, welche in Abbildung 4.12 dargestellt ist, realisiert die Informationsermittlung.
Zunächst wird das Element ermittelt, in welches die einzelnen Kurse geschrieben werden sollen. Anschließend wird überprüft, ob der Nutzer bereits in ein oder mehr Kurse eingeschrieben ist. Ist dies der Fall, wird der Text „No courses entered“ gelöscht, andernfalls wird dem Nutzer dieser Text angezeigt. Als Nächstes wird für jedes Element, welches in dem lokal gespeicherten Objekt vorhanden ist, ein entsprechender Eintrag angelegt. Dafür werden unterschiedliche Elemente angelegt. Ein Listenelement, ein Linkelement, drei Buttonelemente und ein DIV-Element. Das Listen- und das Linkelement werden kombiniert, um die Verlinkung zu dem Kurs an den angezeigten Titel zu ermöglichen. Die drei Buttonelemente dienen dazu, den privaten oder öffentlichen Schlüssel für den jeweiligen Kurs anzuzeigen beziehungsweise diese Informationen wieder auszublenden. In dem DIV-Element werden die Schlüssel angezeigte. Abschließend werden alle Elemente dem Kursobjekt hinzugefügt und es wird dem nächsten Kurseintrag fortgefahren, sofern dieser vorhanden ist.



(a) **(b)**

Abbildung 4.11: Information-PopUp-Fenster Light-/Darkmode

```
1  var courseContainer = document.getElementById('courses')
2  chrome.storage.local.get("bcam").then((result) => {
3    if (result["bcam"] != undefined) {
4      courseContainer.innerHTML = ''
5
6      result["bcam"].forEach(element => {
7        var li = document.createElement('li')
8        var href = document.createElement('a')
9        var pubbtn = document.createElement('button')
10       var privbtn = document.createElement('button')
11       var hiddeDiv = document.createElement('button')
12       var keyDisplay = document.createElement('div')
13
14       keyDisplay.style.display = "none"
15       pubbtn.innerHTML = "Public Key"
16       pubbtn.addEventListener('click', () => {
17         keyDisplay.style.display = "block"
18         keyDisplay.innerHTML = element[3]
19       })
20       privbtn.innerHTML = "Private Key"
21       privbtn.addEventListener('click', () => {
22         keyDisplay.style.display = "block"
23         keyDisplay.innerHTML = element[4]
24       })
25
26       hiddeDiv.innerHTML = "Hide Key"
27       hiddeDiv.addEventListener('click', () => {
28         keyDisplay.style.display = "none"
29       })
30       href.href = element[2]
31       href.innerHTML = element[1]
32       href.target = "_blank"
33       li.appendChild(href)
34       courseContainer.appendChild(li)
35       courseContainer.appendChild(pubbtn)
36       courseContainer.appendChild(privbtn)
37       courseContainer.appendChild(hiddeDiv)
38       courseContainer.appendChild(keyDisplay)
39
40     });
41   }
42 });
```

Abbildung 4.12: Information-PopUp-Fenster Quellcode

4.3 WordPress-Plugin

Das WordPress-Plugin hatte in der ursprünglichen Konzeption mehrere Aufgaben, darunter das Weiterreichen des Feedbacks an die Browser Extension und das Verifizieren von dem abgegebenen Feedback durch einen berechtigten Administrator der Seite. Die Funktion der Weiterreichung des Feedbacks konnte über eine Schnittstelle in dem verwendeten Plugin für die Erstellung und Auswertung der Umfragen realisiert werden. Die Verifikation, ob ein Feedback von einem Nutzer aus der Feedbackgruppe stammt, wurde direkt mit der Signatur Erzeugung verknüpft. In dem WordPress-Plugin wird als Hauptaufgabe die Überprüfung der Verknüpfbarkeit von mehreren abgegebenen Feedbacks durch einen Nutzer ausgeführt.
Der Quellcode des Plugins ist im Anhang verfügbar und wird nachfolgend erläutert. Plugins für WordPress werden in PHP geschrieben und besteht in dem vorliegenden Fall aus zwei PHP-Dateien, einer JavaScript-Bibliothek und einer CSS-Datei. Die erste PHP-Datei „bcam-plugin.php“ und ist die Hauptdatei des Plugins, in dieser müssen zu Beginn einige sogenannte „header comments“ angelegt werden, aus denen WordPress die Informationen zu dem Plugin ausliest. In diesem Fall bestehen diese Informationen aus dem Plugin Namen, einer Versionsnummer, einer Lizenzangabe und dem Namen des Autors. Der nächste Schritt besteht darin, eine Funktion zu einem Action Hook mittels „add_action“ hinzuzufügen. Actions werden bei verschiedenen Aktionen ausgeführt, in dem gezeigten Fall ist es das Aufrufen des Backends von WordPress. Beim Aufrufen des Backends wird die Funktion „bcam_plugin_setup_menu()“ ausgeführt. Diese Funktion ist dafür verantwortlich, dass in der Seitenleiste ein Menüeintrag für das Plugin erscheint. Mit dem Befehl „add_menu_page“ wird der Titel, der Pfad zu der anzuzeigenden Seite und ein Icon festgelegt, welches angezeigt werden soll. Die angegebene PHP-Datei „survey_handler.php“ ist für das Anzeigen der Umfragen und das Überprüfen der Ergebnisse auf Verknüpfbarkeit in Bezug auf die Linkable-Ring-Signaturs verantwortlich.
Diese Datei beinhaltet sowohl die Übersicht zu den angelegten Umfragen als auch die Möglichkeit, die Ergebnisse der jeweils ausgewählten Umfrage anzuzeigen. Für die Unterscheidung wird geprüft, ob der POST-Parameter „result“ gesetzt ist, welcher die ID des Kurses beinhalten würde, für welchen die Ergebnisse betrachtet werden sollen. Ist dies nicht der Fall, wird die Funktion „get_surveys“ aufgerufen, diese fragt mittels eines SQL-Statements die Umfragen aus der WordPress-Datenbank des SurveyJS-Plugins ab und erstellt für jede Umfrage einen Eintrag auf der Seite mit einer Schaltfläche, um sich die Ergebnisse anzeigen zu lassen. In dieser Schaltfläche ist die Umfrage ID hinterlegt und beim Betätigen dieser wird diese als POST-Parameter an die Seite gesendet, Abbildung 4.13 verdeutlicht dies.
Ist der Parameter gesetzt, fällt die eingangs beschriebene Überprüfung positiv aus. Es wird zunächst eine CSS-Datei geladen, um das Aussehen der Ergebnisse etwas übersichtlicher zu gestalten, anschließend wird die Bibliothek für die Funktionalität der Linkable-Ring-Signatur geladen. Als Nächstes werden die einzelnen Ergebnisse geladen, welche von den Nutzern übermittelt wurden. Da in diesem Schritt nur die Ergebnisse relevant sind, welche eine Signatur haben, müssen entsprechend alle anderen Ergebnis gefiltert werden, dafür wird überprüft, ob die Signatur vorhanden ist, sonst wird das Ergebnis übersprungen. Anschließend werden die gefilterten Ergebnisse für die tabellarische Darstellung aufbereitet und auf die Seite geschrieben, sodass schlussendlich nur einmalige Ergebnisse angezeigt werden.
Um die Verknüpfbarkeit prüfen zu können, also ob ein Nutzer bereits Feedback zu der ausgewählten Umfrage abgegeben hat, wird mittels JavaScript die gefilterten Elemente aus der Tabelle ermittelt. Anschließend werden die einzelnen Elemente jeweils untereinander und miteinander auf Verknüpfbarkeit geprüft. Dafür wird die entsprechende „link“ Funktion aus der Bibliothek von Victor Taelin

aufgerufen und die erste Signatur mit allen Nachfolgenden verglichen. Das erste Element wird in ein Array gespeichert und falls eine Verknüpfbarkeit festgestellt werden sollte, werden die Einträge aus den zu überprüfenden Ergebnissen entfernt und in ein Array, welches die Duplikate beinhaltet, gespeichert. Dies wird so lange wiederholt, bis alle Ergebnisse betrachtet wurden. Abschließend werden die Einträge, welche Duplikate sind, farblich hervorgehoben. Es werden auch Ergebnisse angezeigt, welche nicht von einer verifizierbaren Person aus der Feedbackgruppe stammen. Dies ist damit begründet, dass potenzielle darin wertvolles Feedback zu finden sein könnte. Möchte der Auswertende sich diese Ergebnisse nicht anzeigen lassen, so findet dieser am Ende der Tabelle zwei Schaltflächen zum Filtern der Ergebnisse. Damit können wahlweise die Duplikate und die nicht validen Ergebnisse gelöscht werden. Als dritte Schaltfläche hat er die Möglichkeit, sich die Ergebnisse als eine CSV-Datei zu exportieren, Abbildung 4.14 stellt diese Funktion dar.

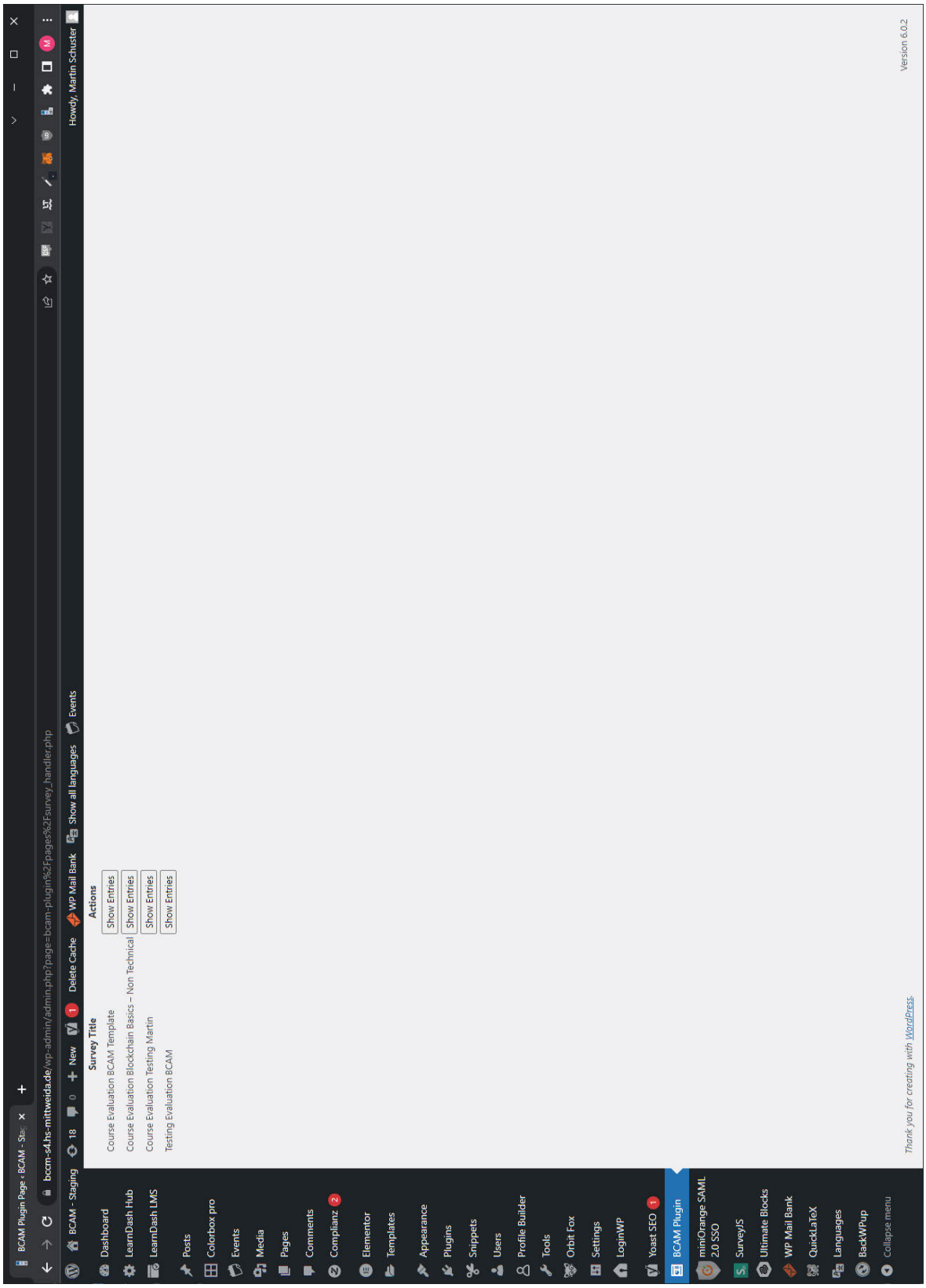


Abbildung 4.13: WordPress-Plugin Umfrage Übersicht

Abbildung 4.14: WordPress-Plugin Umfrageergebnis Übersicht

5 Evaluation

Die Evaluation dient dazu, das aus Kapitel 3 und 4 umgesetzte System vor dem Praxiseinsatz testen zu können. Dazu wurde, wie schon in Kapitel 3 konzipiert, die Evaluation in zwei Bereiche aufgeteilt. Zum einen in eine funktionale Evaluation, bei der die Anforderungen an die Anwendung anhand von Testfällen überprüft wurden. Der zweite Bereich bestand aus einer Erprobung der Anwendung mithilfe einer Probandengruppe, diese sollte einen Ersteindruck liefern, inwieweit die Anwendung bereit ist, in der produktiven Umgebung eingesetzt zu werden.

5.1 Funktionale Evaluation

Die funktionale Evaluation wurde anhand von 12, im Vorfeld definierten, Testfällen durchgeführt, welche verschiedene Situationen abdecken sollten, die während der Laufzeit der Anwendung auftreten können. Die Durchführung der Test wurde mithilfe von drei Browsern (Chrome, Opera und Brave) an einem Rechner sowie an einem weiteren Rechner durchgeführt. Für die Durchführung wurde ein eigener Testkurs mit einer separaten Umfrage angelegt, um mögliche Einflüsse durch vorherige Testläufe ausschließen zu können. Die Testfälle wurden nacheinander abgearbeitet und die Ergebnisse notiert und sind in der Tabelle 5.1 dargelegt. Das System konnte alle 12 Testfälle erfolgreich absolvieren, beziehungsweise trat das zu erwartende Ergebnis ein. Besonders der Umstand, dass Testfall 7 und die Fälle 8, 9 und 10 bestanden wurden, ist für den Produktiveinsatz sehr vorteilhaft. Da in der Vorbetrachtung und mit der Auswahl der Linkable-Ring-Signature Bibliothek noch nicht sicher war, wie genau sich das System verhalten wird, wenn ein neuer Nutzer zur Laufzeit nachträglich in die Feedbackgruppe eintritt. Somit muss keine einschreibe Frist beachtet werden, in der die Nutzer sich in eine Feedbackgruppe einschreiben müssen, sondern das Feedback kann sofort mit Installation der Browser Erweiterung überprüft werden. Eine Problematik, für die noch keine Lösung gefunden werden konnte, ist die, wenn ein Nutzer die Erweiterung absichtlich oder unabsichtlich neu installiert. Dadurch wird der lokale Schlüsselspeicher gelöscht und der Nutzer kann sich neu einschreiben. Wodurch nicht mehr überprüfbar ist, ob er bereits mit einem vorherigen Schlüsselpaar Feedback abgesendet hat. Zudem bleibt der öffentliche Schlüssel in dem Smart Contract gespeichert, sodass ein Weiterreichen der Schlüsselinformationen an einen anderen Nutzer möglich wäre, dieser kann in seinem Namen an Umfragen teilnehmen.

Tabelle 5.1: Auswertung Testfälle

Nummer	Beschreibung	Ergebnis
1	Einschreiben in eine Feedbackgruppe eines Kurses.	Bestanden - Funktion stellte keine Probleme dar.
2	Mehrmaliges einschreiben in eine Feedbackgruppe.	Bestanden - Ist ohne Verwendung eines alternativen Browsers, Rechners oder eine vorherige Neuinstallation nicht möglich.
3	Absenden eines Feedbacks ohne installierte Browser Erweiterung.	Bestanden - Feedback wurde wie gehabt SurveyJS Plugin gespeichert.
4	Absenden eines Feedbacks mit installierter Browser Erweiterung, aber ohne in die Feedbackgruppe eingeschrieben zu sein.	Bestanden - Feedback wurde mit Signatur abgespeichert und wird als invalides Feedback markiert.
5	Absenden eines Feedbacks mit installierter Browser Erweiterung und erfolgter Einschreibung in die Feedbackgruppe.	Bestanden - Feedback wird signiert und als valides Feedback angezeigt.
6	Wiederholtes abgeben eines Feedbacks von einem Nutzer zum selben Kurs.	Bestanden - Duplikat Kennzeichnung des Feedbacks findet statt.
7	Ein weiterer Nutzer schreibt sich in die Feedbackgruppe ein.	Bestanden - Funktionierte ohne Probleme.
8	Nutzer 1 gibt ein Feedback ab.	Bestanden - Feedback wird korrekt aufgenommen und als Duplikat markiert.
9	Nutzer 2 gibt ein Feedback ab.	Bestanden - Feedback wird signiert und ist ein neues Feedback eines Nutzers.
10	Nutzer 2 gibt erneut ein Feedback ab.	Bestanden - Feedback wird als Duplikat erkannt.
11	De- und anschließende Neuinstallation der Erweiterung.	Bestanden - Nutzer kann sich erneut in die Feedbackgruppe einschreiben.
12	Nach Neuinstallation und erneutem Einschreiben Abgabe eines Feedbacks.	Bestanden - Feedback wird signiert und ist ein neues Feedback eines Nutzers.

5.2 Evaluation mit Probandengruppe

Der zweite Teil der Evaluation bestand aus einem Test mit einer Probandengruppe. Dabei wurde hauptsächlich die Komponente der Browser Extension von der Probandengruppe betrachtet, da dies die einzige Komponente ist, mit der der Nutzer später direkt interagiert. Es wurde versucht, eine möglichst diverse Probandengruppe im Hinblick auf Alter, Geschlecht, Bildungsstand und technische Vorerfahrungen zu befragen, um mögliche Demografien ermittelt zu können, bei welchen das System auf besonderen Zuspruch stößt oder bei denen das System eher abgelehnt wird. Nach der Evaluation wurden die ausgefüllten Fragebögen ausgewerteten und die Erkenntnisse sollen in die weitere Entwicklung des Systems einfließen. Damit sollen eventuelle Fehler und Unklarheiten im Vorfeld beseitigt werden, um für einen möglichen Praxistest im Wintersemester 2022/2023 optimal vorbereitet zu sein.

5.2.1 Testablauf

Die Evaluation mit den Probanden wurde vorrangig online durchgeführt, da es sich um ein Onlinesystem handelt und die Probanden die Browser Erweiterung auf ihren gewohnten Systemen nutzen sollten, um mögliche Inkompatibilitäten mit Software oder Hardware ermitteln zu können. Für eine kurze Einführung in die Thematik und eventuelle Fragen stand der Autor den Probanden über eine Zoom-Session beziehungsweise Discord Voice Chat zur Verfügung.[32][33] Nach der Einführung wurden die Probanden auf das Testsystem geleitet, wo sie wie der spätere Nutzer die Möglichkeit hatten, die Browser Erweiterung herunterzuladen und zu installieren. Anschließend haben sie sich in eine Feedbackgruppe eines Testkurses eingeschrieben und beispielhaft eine Umfrage ausgefüllt. Abschließend wurden sie gebeten, einen Fragebogen auszufüllen und entsprechend demografische Angaben zu tätigen sowie das Erlebte zu bewerten und zu kommentieren. Für jede Interviewsession wurde ein Zeitfenster von 30 Minuten vorgesehen.

5.2.2 Auswertung Ergebnisdiskussion

Die Evaluation wurde mit 18 Probanden durchgeführt. Wobei 55,6% der Teilnehmer männlich, 33,3% weiblich und 11,1% diverse Teilnehmer waren. Mit 66,7% war die Altersgruppe der 21- bis 29-Jährigen am stärksten vertreten, gefolgt von den 30- bis 39-Jährigen mit 11,1% und die Altersgruppen der 18 bis 20, 40 bis 49, 50 bis 59 und 60 oder älteren waren jeweils mit 5,6% in der Probandengruppe vertreten. 33,3% der Probanden verfügten über einen Bachelor Abschluss, 27,8% besitzen ein Abitur oder gleichwertigen Abschluss, 22,2% über einen Masterabschluss und 16,7% studieren aktuell. Damit entsprechen die demografischen Daten der erhofften Zielgruppe für die Evaluation und die Daten sind entsprechend in Abbildung 5.1 aufgezeigt.
Die Auswertung der Daten zu der Anwendung ergab, dass 61,1% der Probanden den Chrome-Browser benutzten, 16,7% den Opera Browser, 11,1% Brave Browser, Vivaldi und Edge waren jeweils mit 5,6% vertreten. 11,1% der Probanden hatten Problemen während der Installation. Diese ließen sich nicht eindeutig identifizieren, eine kurzfristige Abhilfe schaffte das Verwenden eines anderen geeigneten Browsers, eine genaue Betrachtung, was die Probleme bei Installation der Erweiterung verursachte, ist vor der Veröffentlichung geplant. Für den produktiven Einsatz sollte eine Veröffentlichung im Chrome-Web-Store in Betracht gezogen werden. Damit die Installation nicht immer über den Entwicklermodus geschehen muss, wäre die Möglichkeit, die Erweiterung aus dem

Chrome Web Store beziehen zu können, eine für den Nutzer komfortable Lösung. Es muss dahingehend geprüft werden, ob die Erweiterung den Vorgaben des Chrome-Web-Stores entspricht und gegebenenfalls angepasst werden. In Abbildung 5.2 sind die entsprechenden Daten dargestellt.
Der Großteil der Probandengruppe (66,7%) empfand das minimale Design als ausreichend bis optimal, für 33,3% der Nutzer hätte die Browser Erweiterung mehr Feedback geben können, was die Erweiterung zum aktuellen Zeitpunkt tätigt. Als Beispiel wurde das Einblenden eines Icons im Symbol der Browser Extension genannt, womit signalisiert wird, dass die Signatur erstellt und gespeichert wurde. Durch das minimale Design war das Vertrauen in die Anwendung bei einem Probanden nicht gegeben und dieser hätte sich aufgrund dessen gegen die Anwendung entschieden. Generell würden jedoch 77,8% der Probanden diese Lösung bevorzugen gegenüber einem Codewort basierenden System. Dabei würden die Nutzer ein einmal zu verwendendes Codewort erhalten, um eine Umfrage auszufüllen. Es muss jedoch bei diesem System darauf vertraut werden, dass die Ergebnisse und das Codewort nicht zusammen gespeichert werden Abbildung 5.3 verdeutlicht die Daten.
Von den Probanden wurde sich abschließend gewünscht, eine Umsetzung für mehr Browser, beispielsweise Firefox anzubieten. Es muss entsprechend geprüft werden, wie hoch der Aufwand ist, die erarbeitete Lösung an andere Browser Engines anzupassen (Firefox, Safari). Zudem wurde erfragt, ob solch eine Lösung auch auf mobilen Endgeräten möglich wäre, dies ist ebenfalls zu prüfen. Insgesamt konnte für die präsentierte Lösung ein größtenteils positives Feedback aufgenommen werden, was weitere Arbeiten und den praktischen Einsatz rechtfertigen. Die beiliegende Datei „Evaluation dezentrales Feedback-System.xlsx“ beinhaltet die Ergebnisse der Umfrage in tabellarischer Form.

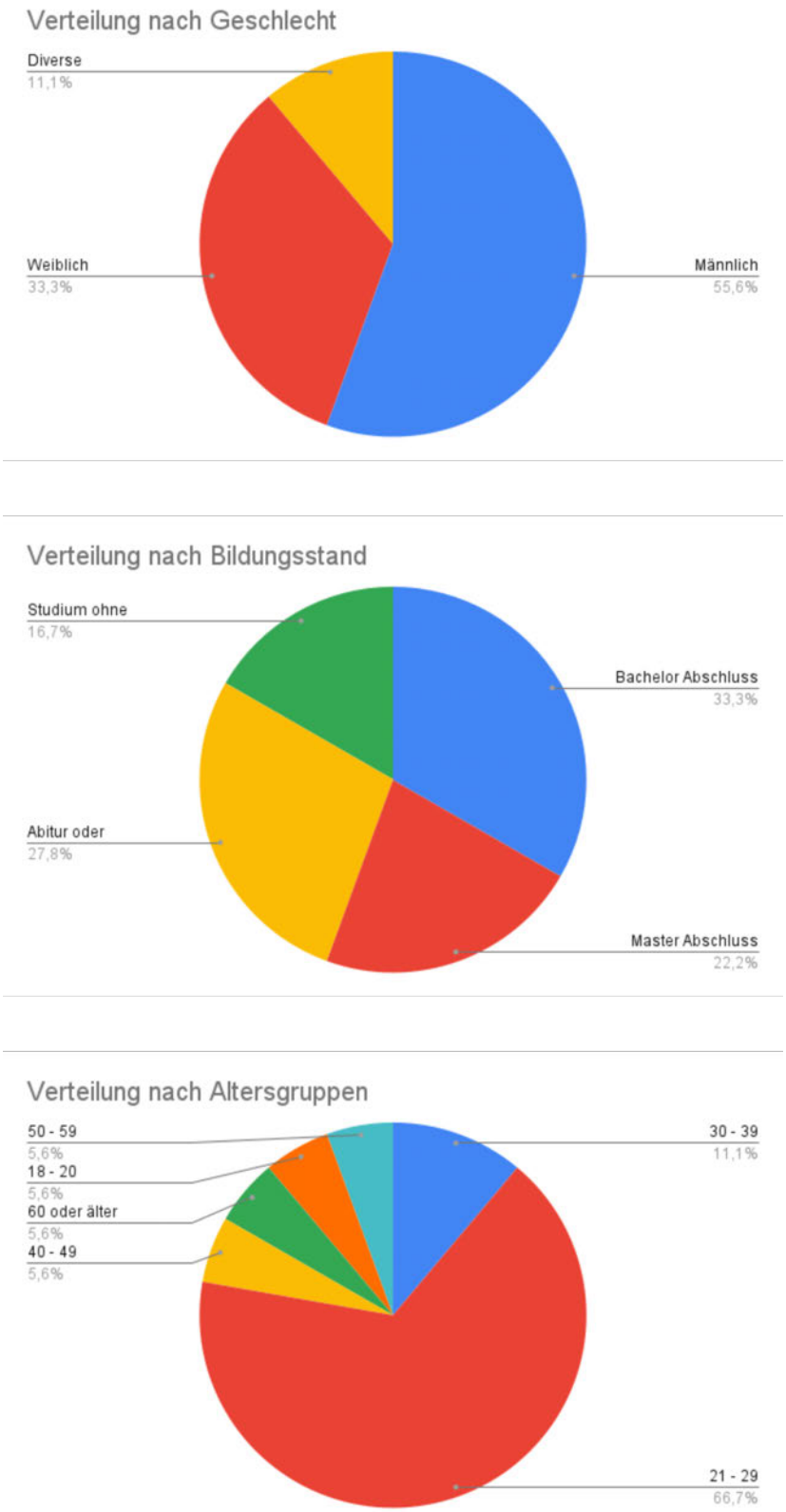


Abbildung 5.1: Auswertung demografische Angaben

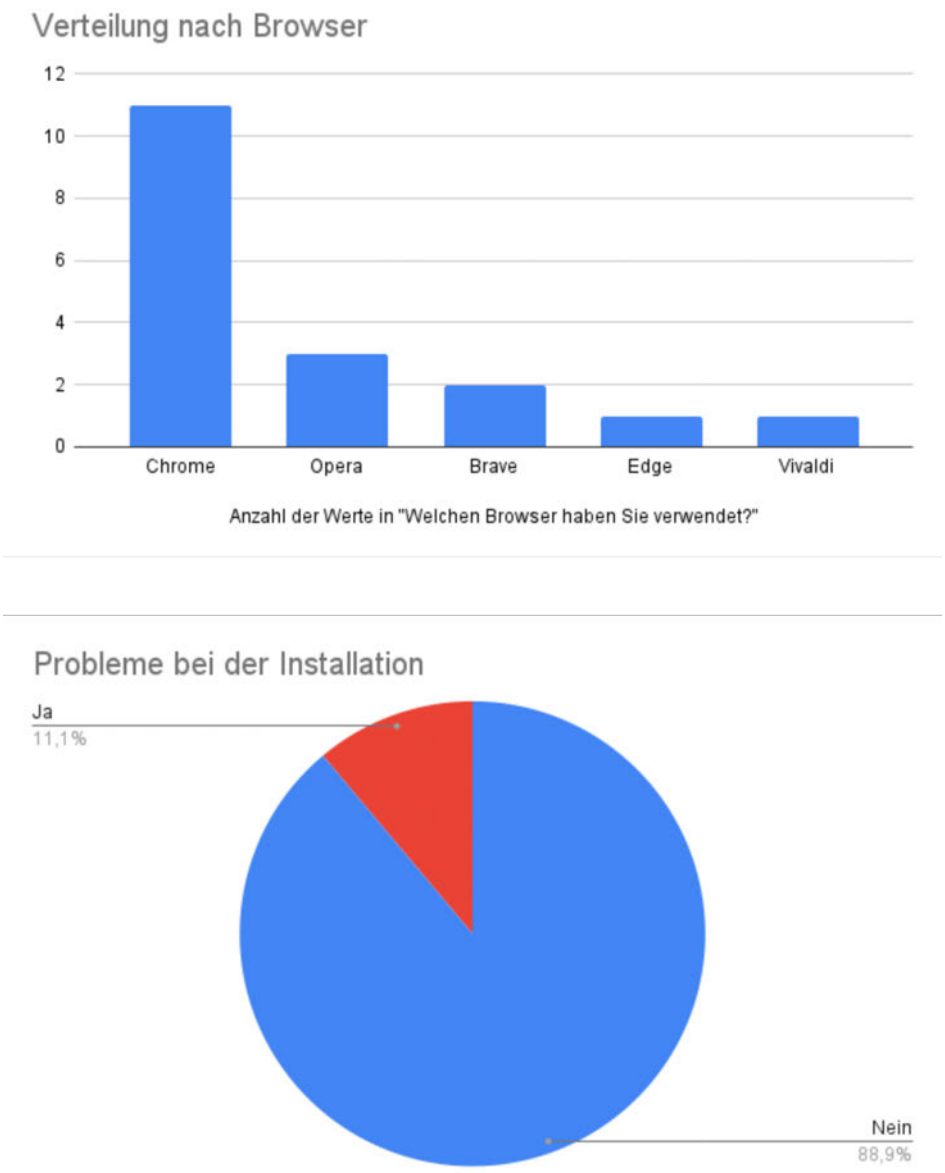


Abbildung 5.2: Auswertung technische Angaben

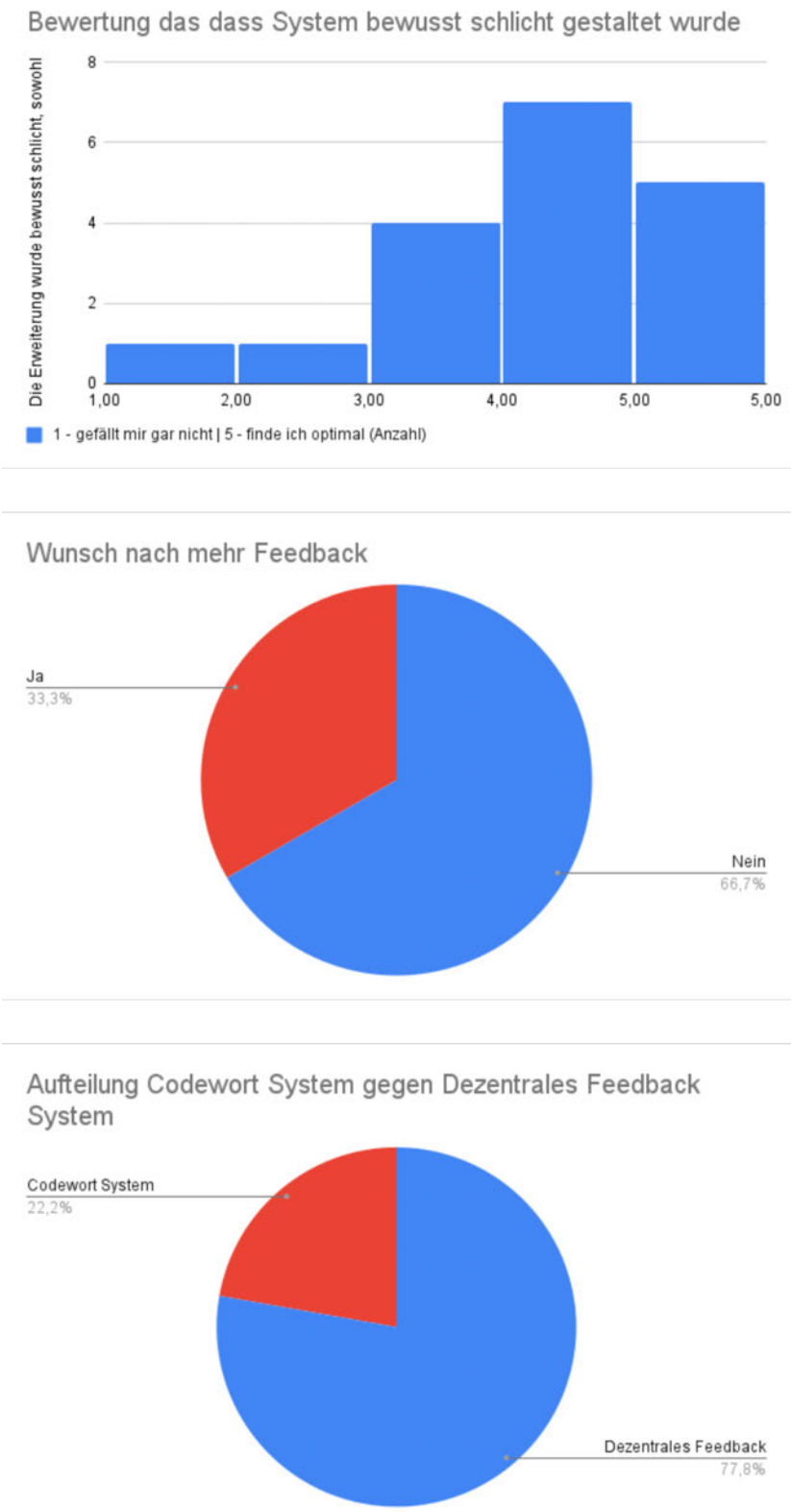


Abbildung 5.3: Auswertung Bewertung Browser Extension

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung

Mit der vorliegenden Arbeit konnte gezeigt werden, dass ein dezentrales Feedback-System für die Blockchain Academy Mittweida, von der Konzeption bis hin zur Umsetzung, erarbeitet werden konnte. Ein funktionaler Test anhand von 12 Testfällen ergab, dass die Anwendung diese Testfälle in der erwarteten Art und Weise bestanden hat. Die Ergebnisse aus einem ersten Test mit einer Probandengruppe sprechen dafür, dass dieses System eine Lösung für die eingangs erläuterte Problematik der Anonymität im Netz sein kann. Insgesamt wurde das System sehr gut aufgenommen und die Probanden würden dieses mehrheitlich gegenüber einem System mit einem einmal Token oder Codewort zur Verifizierung bevorzugen.
Während der Entwicklung gab es die meisten Probleme bei der Umsetzung der Browser Erweiterung. Dies war unerwartet, da die im Vorfeld getätigte Recherche, wie diese entwickelt werden ergab, dass Browser Erweiterungen ähnlich der Web-Entwicklung sind und hauptsächlich aus HTML, Javascript und CSS Bestandteilen besteht. Da bereits einiges an Vorerfahrung auf diesem Gebiet bestand, Spezialisierung im Studium sowie die Web-Entwicklung Teil der Aufgaben aus der Anstellung im Projekt der Blockchain Academy Mittweida sind. Wurde ein wesentlich kürzer Zeitraum für die Entwicklung angedacht. Jedoch zeigte sich bei der Entwicklung, dass viele der Herangehensweisen aufgrund von Sicherheitsbeschränkungen nicht wie bereits bekannt und erprobt umzusetzen sind. Als Beispiel sei zum einen das Nutzen externen Bibliotheken genannt, in der Web-Entwicklung werden diese als NodeJs-Package installiert oder über ein Content-Delivery-Network (CDN) zu Beginn einer HTML-Datei eingebunden. Die Sicherheitsrichtlinien für Browser Erweiterungen für auf Google Chrome basierte Browser verbietet diese Art des Einbindens jedoch. Die Lösung für dieses Problem ist, die entsprechend benötigten Bibliotheken mittels eines JavaScript-Bundler-Tools wie WebPack oder Browserify für die Verwendung im Browser verfügbar zu machen. Ein weiteres Beispiel ist die Umsetzung der Einschreibung in eine Feedbackgruppe, dafür sollte das Pop-up-Fenster der Browser Extension geöffnet werden bei dem Besuch eines Kurses, in den der Nutzer noch nicht eingeschrieben ist. Abermals war dies durch die Sicherheitsbestimmungen nicht möglich, wodurch eine neue Lösung erdacht werden musste, welche daraus bestand, den entsprechenden Dialog zum Einschreiben in die Seite zur Laufzeit zu injizieren.
Diese und viele kleinere Probleme verzögerten die Fertigstellung der Anwendung. Diese Verzögerungen konnte jedoch einigermaßen durch die vorzeitige Fertigstellung des Smart Contracts abgefangen werden, wodurch eine Beendigung der technischen Umsetzung im Rahmen der vorliegenden Arbeit gewährleistet werden konnte. Das finale Design der Anwendung sowohl im Front-End Bereich der Browser Erweiterung als auch im Backend des WordPress-Plugin konnte aufgrund der erwähnten Verzögerung noch nicht abgeschlossen werden.

6.2 Ausblick

Der erste angelegte Praxistest neben der Probandengruppe für die Evaluation für das System ist für den Beginn des Wintersemesters 2022/2023 im Masterstudiengang Blockchain & Distributed Ledger Technologies (DLT) angedacht. In diesem Studiengang sind die Kurse auf der Blockchain Academy

Mittweida Teil der Lehrmaterialien. Dadurch ist es möglich, über ein Semester hinweg mit der Browser Erweiterung verifizierbares Feedback zu sammeln. Dabei würde zu Beginn des Semesters den Studierenden die Erweiterung präsentiert werden, wie diese funktioniert und wie diese zu benutzen ist. In regelmäßigen Abständen wird kontrolliert, ob das System noch ordnungsgemäß funktioniert und am Ende des Semesters werden die Studierenden gebeten, eine Umfrage zu ihrer Erfahrung mit der Erweiterung auszufüllen. Durch die Umstellung der Ethereum-Blockchain auf den Proof-of-Stake-Konsens wird das Rinkeby Testnetz voraussichtliche im dritten Quartal 2023 nicht weiter betrieben. Aufgrund dessen muss der Smart Contract, zur Speicherung der öffentlichen Schlüssel, auf eines der neuen Testnetze oder das Mainnet portiert und gegebenenfalls angepasst werden.
Weitere Arbeiten, die an dem gesamten System noch erfolgen sollen, sind eine Anpassung des Designs und der User-Experience (UX). Da während der Entwicklung das Hauptaugenmerk darauf lag, die Funktionalität umzusetzen, wurden diese Aspekte des Designs und der UX zunächst vernachlässigt. Sollte sich die Anwendung im Praxistest behaupten, wäre es denkbar, diese auch weiterzuverbreiten, dafür wäre es notwendig, die Modularität zu verbessern, da momentan die Funktionsweise des Feedbacks abfragen und speichern auf das SurveyJS-Plugin angepasst ist. Ein universeller Ansatz, sodass dieses System für die meisten Feedback-Tools möglich ist, wäre durch das Schaffen von fest definierten Schnittstellen möglich. Es wäre auch denkbar, eine Lösung mit einem eigenen Feedback-Tool anzubieten. Damit wäre die gesamte Softwarelösung unabhängig von Veränderungen durch Drittanbietern.

Anhang A: Umfrageformular

Evaluation dezentrales Feedback-System

Mit diesem Fragebogen soll das entwickelte dezentrale Feedback-System für die Blockchain Academy Mittweida erprobt werden und auf seine Praxistauglichkeit überprüft werden. Zunächst werden einige demografische Daten erfasst, welche helfen sollen, die Antworten besser in einen Kontext bringen zu können.

Geschlecht

- ○ Weiblich
- ○ Männlich
- ○ Diverse

Alter

- ○ 17 oder jünger
- ○ 18 - 20
- ○ 21 - 29
- ○ 30 - 39
- ○ 40 - 49
- ○ 50 - 59
- ○ 60 oder älter

Bildungsstand

- ◯ kein Schulabschluss
- ◯ Unterer Schulabschluss
- ◯ Abitur oder gleichwertiger Abschluss
- ◯ Studium ohne Abschluss
- ◯ Bachelor Abschluss
- ◯ Master Abschluss
- ◯ Doktor Grad

Sofern Sie nicht bereits einen Browser mit der Chrome Engine benutzen, laden Sie sich bitte solch einen Browser herunter und öffnen Sie anschließend den folgenden Link unter, welchem Sie die Browser Erweiterung finden können und installieren Sie diese. Auf der Seite gehen Sie anschließend auf die Kurs-Übersichtseite und öffnen dort den Kurs Browser Extension Testing Course und tragen sich über die "Enrol in Course" Schaltfläche in das Feedback-System ein. Öffnen Sie anschließend die Beispielumfrage Survey for Browser Extension Testing und füllen diese aus. Kehren Sie nach dem Ausfüllen und Abschicken der Umfrage hier her zurück.

Welchen Browser haben Sie verwendet?

Meine Antwort

Hatten Sie Probleme bei der Installation der Erweiterung?

- ◯ Ja
- ◯ Nein

Falls Sie Probleme hatte, bitte erläutern Sie kurz welche aufgetreten sind:

Meine Antwort

Die Erweiterung wurde bewusst schlicht, sowohl vom Aussehen als auch von der Funktion, der Großteil der Funktion findet unbemerkt vom Nutzer statt, gestaltet, wie finden Sie das?

	1	2	3	4	5	
Gefällt mir garnicht	○	○	○	○	○	Finde ich optimal

Würden Sie sich mehr Feedback durch die Anwendung wünschen?

- ○ Ja
- ○ Nein

Falls Ja, in welcher Form soll das Feedback erfolgen und was genau stellen Sie sich vor:

Meine Antwort

Hätten Sie die Wahl für ein Codewort basiertes Umfragesystem, bei dem Sie einen individuellen Code erhalten um an der Umfrage Teilzunehmen oder diesem System Welches würden Sie bevorzugen?

- ○ Codewort System
- ○ Das vorgestellte System

Haben Sie noch fragen oder Anregungen, lass Sie mich diese gerne wissen:

Meine Antwort

Vielen Dank für Ihre Teilnahme an der Evaluation!

Senden

Alle Eingaben löschen

Literaturverzeichnis

[1] Ronald L. Rivest, Adi Shamir und Yael Tauman. „How to Leak a Secret“. In: *Advances in Cryptology — ASIACRYPT 2001*. Hrsg. von Colin Boyd. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, S. 552–565. ISBN: 978-3-540-45682-7.

[2] Eiichiro Fujisaki und Koutarou Suzuki. „Traceable Ring Signature“. In: *Public Key Cryptography – PKC 2007*. Hrsg. von Tatsuaki Okamoto und Xiaoyun Wang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, S. 181–200. ISBN: 978-3-540-71677-8.

[3] William J. Buchanan. *Ring Signatures*. 2021. URL: https://asecuritysite.com/encryption/ring_sig.

[4] Joseph K. Liu und Duncan S. Wong. „Linkable Ring Signatures: Security Models and New Schemes“. In: *Computational Science and Its Applications – ICCSA 2005*. Hrsg. von Osvaldo Gervasi u. a. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, S. 614–623. ISBN: 978-3-540-32044-9.

[5] Emmanuel Bresson, Jacques Stern und Michael Szydlo. „Threshold Ring Signatures and Applications to Ad-hoc Groups“. In: *Advances in Cryptology — CRYPTO 2002*. Hrsg. von Moti Yung. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, S. 465–480. ISBN: 978-3-540-45708-4.

[6] Joseph K. Liu, Victor K. Wei und Duncan S. Wong. „Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups“. In: *Information Security and Privacy*. Hrsg. von Huaxiong Wang, Josef Pieprzyk und Vijay Varadharajan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, S. 325–335. ISBN: 978-3-540-27800-9.

[7] Satoshi Nakamoto. „Bitcoin: A Peer-to-Peer Electronic Cash System“. In: (Mai 2009). URL: http://www.bitcoin.org/bitcoin.pdf.

[8] Vitalik Buterin. „A NEXT GENERATION SMART CONTRACT & DECENTRALIZED APPLICATION PLATFORM“. In: 2015.

[9] Klint Finley. *A 50 million hack just showed that the DAO was all too human.* 2016. URL: https://www.wired.com/2016/06/50-million-hack-just-showed-dao-human/#:~:text=4%3A30%20AM-,A%20%2450%20Million%20Hack%20Just%20Showed%20That%20the%20DAO%20Was,take%20out%20of%20the%20equation..

[10] Chris Dannen. „Introducing Ethereum and Solidity“. In: Apress Berkeley, CA, 2017. ISBN: 978-1-4842-2534-9. DOI: 10.1007/978-1-4842-2535-6. URL: https://doi.org/10.1007/978-1-4842-2535-6.

[11] IONOS Inc. *Asymmetric encryption: using public key encryption to ensure secure data transfer.* 2022. URL: https://www.ionos.com/digitalguide/server/security/public-key-encryption/.

[12] Jens Kubieziel. „Symmetrische und asymmetrische Verfahren der Kryptographie“. In: ().

[13] Shafaq Naheed Khan u. a. „Blockchain smart contracts: Applications, challenges, and future trends“. In: *Peer-to-Peer Networking and Applications* 14.5 (2021), S. 2901–2925. ISSN: 1936-6450. DOI: 10.1007/s12083-021-01127-0.

[14] Wissal Haji. *Learn Solidity: The Factory Pattern*. 2020. URL: https://betterprogramming.pub/learn-solidity-the-factory-pattern-75d11c3e7d29.

[15] Joe Messerman Peter Murray Nate Welch. *EIP-1167: Minimal Proxy Contract*. 2018. URL: https://eips.ethereum.org/EIPS/eip-1167.

[16] Mario Oettler. *Factory/Clone*. 2021. URL: https://blockchain-academy.hs-mittweida.de/courses/solidity-coding-beginners-to-intermediate/lessons/solidity-11-coding-patterns/topic/factory-clone/.

[17] Abdullah A. Zarir u. a. „Developing Cost-Effective Blockchain-Powered Applications: A Case Study of the Gas Usage of Smart Contract Transactions in the Ethereum Blockchain Platform“. In: *ACM Trans. Softw. Eng. Methodol.* 30.3 (März 2021). ISSN: 1049-331X. DOI: 10.1145/3431726. URL: https://doi.org/10.1145/3431726.

[18] Gavin Wood u. a. „Ethereum: A secure decentralised generalised transaction ledger“. In: *Ethereum project yellow paper* 151.2014 (2014), S. 1–32.

[19] Kev Zettler. *Was ist ein verteiltes System?* 2022. URL: https://www.atlassian.com/de/microservices/microservices-architecture/distributed-architecture.

[20] cleanpng.com. *Dezentralisierung die Dezentrale system-Zentralisierung Dezentraler Planung - andere*. 2022. URL: https://de.cleanpng.com/png-xkpm1b/.

[21] Martin Rost. „Über die Funktionalität von Anonymität für die bürgerliche Gesellschaft“. In: *Anonymität im Internet: Grundlagen, Methoden und Tools zur Realisierung eines Grundrechts*. Hrsg. von Helmut Bäumler und Albert von Mutius. Wiesbaden: Vieweg+Teubner Verlag, 2003, S. 62–73. ISBN: 978-3-663-05790-1. DOI: 10.1007/978-3-663-05790-1_6. URL: https://doi.org/10.1007/978-3-663-05790-1_6.

[22] web3js.org. *web3.js - Ethereum JavaScript API*. 2022. URL: https://web3js.readthedocs.io/en/v1.8.0/.

[23] Chrome Developers. *Welcome to Manifest V3*. 2022. URL: https://developer.chrome.com/docs/extensions/mv3/intro/.

[24] WordPress-Foundation. *WordPress*. 2022. URL: https://wordpress.org/.

[25] Devsoft Baltic OÜ.

[26] Learndash. *LearnDash*. 2022. URL: https://www.learndash.com/.

[27] SoftSelect-GmbH. *SoftSelect Anforderungskatalog*. 2022. URL: http://www.softselect.de/business-software-glossar/anforderungskatalog#:~:text=Ein%20Anforderungskatalog%20ist%20die%20detaillierte,das%20betreffende%20Softwaresystem%20leisten%20muss..

[28] Google. *Google Formulare: App zum Erstellen von Onlineformularen*. 2022. URL: https://www.google.de/intl/de/forms/about/.

[29] Remix Project. *Remix IDE*. Version 0.26.3. 30. Sep. 2022. URL: https://remix-project.org/.

[30] Infura Inc. *Infura*. 2022. URL: https://infura.io/.

[31] Victor Taelin. *Linkable Ring Signatures on JavaScript and PureScript*. Version 0.1.5. 20. Sep. 2016. URL: https://github.com/VictorTaelin/lrs.

[32] Zoom Video Communications, Inc. *Zoom*. Version 5.11.4. URL: https://zoom.us/.

[33] Discord Inc. *Discord*. Version 67030. 15. Sep. 2020. URL: https://discord.com.

Eidesstattliche Erklärung

Hiermit versichere ich – Martin Schuster – an Eides statt, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt oder anderweitig veröffentlicht.

Mittweida, 30. September 2022 | Martin Schuster, B.Sc.

Ort, Datum