HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

MASTERARBEIT

Herr
Lukas Thiede

Theoretische Untersuchung der Implementierung eines NFT Marktplatzes auf Ethereums Skalierungslösungen Loopring und Taiko

Technologische Analyse, Entwicklungsaufwand und Vergleich

2024

Fakultät **Angewandte Computer- und Biowissenschaften**

MASTERARBEIT

Theoretische Untersuchung der Implementierung eines NFT Marktplatzes auf Ethereums Skalierungslösungen Loopring und Taiko

Technologische Analyse, Entwicklungsaufwand und Vergleich

Autor:
Lukas Thiede

Studiengang:
Blockchain & Distributed Ledger Technologie

Seminargruppe:
BC22w1-M

Erstprüfer:
Prof. Dr.-Ing. Andreas Ittner

Zweitprüfer:
Dipl. Volkswirt Mario Oettler

Mittweida, Dezember 2024

Bibliografische Angaben

Thiede, Lukas: Theoretische Untersuchung der Implementierung eines NFT Marktplatzes auf Ethereums Skalierungslösungen Loopring und Taiko, Technologische Analyse, Entwicklungsaufwand und Vergleich, 102 Seiten, 10 Abbildungen, Hochschule Mittweida, University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften

Englischer Titel: *Theoretical examination of implementing an NFT marketplace on Ethereums scaling solutions Loopring and Taiko: technological analysis, development effort and comparison*

Masterarbeit, 2024

Kurzbeschreibung

Diese Masterarbeit analysiert zwei Skalierungslösungen für Ethereum: Loopring, einen anwendungsspezifischen ZK-Rollup, und Taiko, einen dezentralen Rollup mit zkEVM-Funktionalität vom Typ 1 (ethereum-äquivalent). Der Fokus liegt auf den technischen Infrastrukturen, Beweis- und Sicherheitsmechanismen sowie der Eignung als Plattformen für NFT-Marktplätze. Die beiden Rollups werden anhand der Aspekte des Blockchain-Trilemmas Dezentralisierung, Sicherheit und Skalierbarkeit verglichen. Zusätzlich werden die Entwicklungsumgebungen untersucht und auf ihre Entwicklerfreundlichkeit überprüft, einschließlich der Unterstützung für Smart Contracts, der Kompatibilität mit bestehenden Ethereum-Tools und der Flexibilität für verschiedene Anwendungsmöglichkeiten.

Die Analyse zeigt, dass Loopring primär als dApp ausgelegt ist und vor allem vordefinierte Funktionen für NFT-Marktplätze bietet. Es überzeugt durch niedrige Kosten dank des Counterfactual NFT-Ansatzes, ist jedoch zentralisiert und bietet nur eingeschränkte Anpassungsmöglichkeiten für verschiedene Token-Standards oder benutzerdefinierte Features. Taiko hingegen fokussiert sich auf Dezentralität und Sicherheit, mit einem innovativen Inception-Layer-Ansatz zur Skalierung durch die Verschachtelung mehrerer Rollup-Schichten. Zudem ist Taiko durch sein flexibles Design besser darauf ausgelegt, zukünftige Entwicklungen in der Zero-Knowledge-Technologie zu integrieren.

Die Ergebnisse legen nahe, dass Loopring als schnelle und kostengünstige Lösung für grundlegende NFT-Marktplatz-Funktionen wie Minting, Handel und Verkauf geeignet ist. Taiko bietet hingegen eine Plattform für die flexible Entwicklung komplexerer und individueller Lösungen, die Dezentralität priorisieren.

I. Inhaltsverzeichnis

II. Abbildungsverzeichnis

III. Tabellenverzeichnis

IV. Abkürzungsverzeichnis

ABI	Application Binary Interface
AMM	Automated Market Maker
API	Application Programming Interface
BCR	Based Contestable Rollup
BIP	Basispunkt
CID	Content Identifier
DAO	Dezentrale Autonome Organisation
dApp	Dezentrale Anwendung
DeFi	Decentralized Finance
DEX	Dezentrale Kryptobörse (Exchange)
EdDSA	Edwards-Curve Digital Signature Algorithm
EIP	Ethereum Improvement Proposal
EOA	Externally Owned Account
ERC	Ethereum Request for Comment
EVM	Ethereum Virtual Machine
ICO	Initial Coin Offering
IPFS	InterPlanetary File System
L1	Ethereum Layer 1, Main-Chain
L2	Layer 2 Skalierungslösung
LRC	Loopring Coin
NFT	Nicht-fungible Token
OCDA	On-Chain Data Availability
PoS	Proof of Stake
PoW	Proof of Work
RPC	Remote Procedure Call
SDK	Software Development Kit
SGX	Software Guard Extensions
TEE	Trusted Execution Environment
TPD	Transaktionen per Day
TPS	Transaktionen pro Sekunde
URI	Uniform-Resource-Identifier
ZK	Zero-Knowledge

zkEVM ..... Zero Knowledge Ethereum Virtual Machine

ZKP ..... Zero Knowledge Proof

zkSNARK ..... Zero-Knowledge Succinct Non-Interactive Argument of Knowledge

zkVM ..... Zero Knowledge Virtual Machine

Code-Snippet-Verzeichnis

VI. Danksagung

Mein besonderer Dank gilt meinen Eltern, die mir nicht nur einen akademischen Werdegang ermöglicht haben, sondern mich auch während der gesamten Zeit mit ihrer unermüdlichen Unterstützung begleitet haben. Ihre Ermutigung und ihr Rückhalt sind für mich von unschätzbarem Wert.

Ebenso möchte ich Tai Nguyen Nhan meinen aufrichtigen Dank aussprechen. Die fachlichen Diskussionen mit ihm waren stets eine Quelle der Inspiration und haben wesentlich zum Gelingen dieser Arbeit beigetragen. Seine Einsichten und Anregungen haben meine Perspektive erweitert und mich zu neuen Denkansätzen motiviert.

1 Einleitung

Die Entwicklung von Nicht-fungiblen Token (NFTs) hat digitales Besitztum revolutioniert und innovative Möglichkeiten für eine Tokenisierung von Vermögenswerten geschaffen. Gleichzeitig zeigte der rapide Anstieg der Popularität von NFTs eindeutige Skalierungsprobleme der Ethereum-Blockchain. Der NFT-Boom in den Jahren 2020-2022 offenbarte die Grenzen der bestehenden Ethereum-Infrastruktur, die Schwierigkeiten hatte, mit der sprunghaft gestiegenen Nachfrage Schritt zu halten. Dies äußerte sich durch lange Bestätigungszeiten und hohen Transaktionskosten [1].

Eine effiziente Skalierung der Ethereum-Blockchain ist aus mehreren Gründen entscheidend:
Erstens ermöglicht sie eine breitere Zugänglichkeit, indem reduzierte Transaktionskosten einer größeren Nutzerbasis die Teilnahme am NFT-Markt erleichtern. Zweitens eröffnet sie neue Anwendungsmöglichkeiten, indem die Tokenisierung verschiedener digitaler und realer Vermögenswerte zu wettbewerbsfähigen Preisen möglich wird. Drittens könnte eine verbesserte Skalierbarkeit einen erneuten Aufschwung des NFT-Marktes auslösen und innovative Geschäftsmodelle ermöglichen, die bisher aufgrund der hohen Transaktionskosten nicht realisierbar waren.

Rollups bieten vielversprechende Ansätze zur Bewältigung der Skalierungsherausforderungen von Ethereum. Diese Arbeit konzentriert sich auf zwei spezifische Lösungen: Loopring und Taiko, die beide vom selben Gründerteam stammen, jedoch unterschiedliche Ansätze verfolgen. Es werden die technischen Infrastrukturen, Beweismechanismen und Entwicklungsumgebungen dieser Rollups untersucht, um zu betrachten, auf welcher Skalierungslösung heutzutage ein NFT Marktplatz implementiert werden sollte.

2 Methodik

Im Kontext von Blockchain-Technologie ist die Betrachtung des Blockchain-Trilemmas besonders relevant, da es die Herausforderungen beschreibt, die bei der Entwicklung von dezentralisierten Systemen auftreten. Das Trilemma umfasst die drei Eigenschaften **Dezentralisierung**, **Sicherheit** und **Skalierbarkeit**, die oft in Konflikt zueinander stehen. Als Erweiterungen der Ethereum-Blockchain müssen Loopring und Taiko dieses Trilemma adressieren.
Für diese Arbeit werden die Aspekte des Trilemmas betrachtet, um die beiden Skalierungslösungen anhand dieser drei Dimensionen zu vergleichen. Darüber hinaus wird eine vierte Dimension, die **Entwicklerfreundlichkeit**, hinzugefügt, um zu untersuchen, wie zugänglich und benutzerfreundlich die jeweiligen Plattformen für Entwickler sind. Diese Erweiterung ist entscheidend, da eine hohe Entwicklerfreundlichkeit die Adaption und Implementierung von Anwendungen auf diesen Plattformen erleichtert, die Nutzeranzahl auf diesen Plattformen erhöht und somit einen wichtigen Faktor für den Erfolg eines NFT-Marktplatzes darstellt.

Bewertungskriterien
Folgende Kriterien werden für die Analyse von Loopring und Taiko herangezogen:

1. **Sicherheit**
   - Konsensus-Mechanismus bzw. Sicherheitsmodell
   - Implementierung von Sicherheitsaudits und Best Practices
   - Open-Source-Praxis
   - Auditierung der Protokolle

2. **Dezentralisierungsgrad**
   - Verteilung der Validatoren/Nodes
   - Governance-Struktur und Entscheidungsfindungsprozesse
   - Abhängigkeit von zentralisierten Komponenten

3. **Skalierbarkeit**
   - Transaktionen pro Sekunde (TPS)
   - Gaskosten für NFT Minten
   - Maximale Netzwerkkapazität
   - zukunftsorientierter Skalierungsansatz

4. **Entwicklerfreundlichkeit**
   - Qualität und Vollständigkeit der Dokumentation
   - Vertrautheit der Entwicklungsumbegung
   - Verfügbarkeit von Entwicklertools und Software Development Kits (SDKs)

Datenerhebung
Die Datenerhebung erfolgt durch eine Kombination aus:
- Literaturrecherche: Analyse von Whitepapern, wissenschaftlichen Publikationen, Veröffentlichungen der Rollup-Mitarbeiter und den jeweiligen Dokumentationen
- empirische Untersuchung: Erstellung von Konzepten für Prototypen auf beiden Plattformen anhand der Dokumentationen und etablierten Entwicklertools

Gegenüberstellung
Die gesammelten Daten werden in einer Tabelle gegenübergestellt. Anschließend folgt eine kurze Bewertung.

Abschließende Bewertung
Basierend auf den Ergebnissen der Gegenüberstellung wird eine differenzierte Bewertung für die besser geeignete Skalierungslösung für ein neues NFT-Marktplatz-Projekt formuliert, dabei werden die spezifischen Anforderungen und Prioritäten eines solchen Projekts berücksichtigt.

3 Die Ethereum-Blockchain

Das folgende Kapitel bietet einen Überblick über Ethereum und seine Skalierungsherausforderungen. Es dient als Grundlage für das Verständnis der aktuellen Entwicklungen im Ethereum-Ökosystem und der verschiedenen Lösungsansätze zur Bewältigung der Skalierungsprobleme.

3.1 Technische Grundlagen

Ethereum basiert auf einer verteilten Ledger-Technologie, die alle Transaktionen in einem öffentlichen, unveränderlichen Ledger aufzeichnet. Die Ethereum Virtual Machine (EVM) führt diese Transaktionen und Smart Contracts aus, die von jedem Knoten im Ethereum-Netzwerk validiert werden. Die Skalierbarkeit von Ethereum Layer 1, der Main-Chain, wird durch folgende Faktoren beeinflusst:

`Blockgröße und Blockzeit` Ethereum hat eine durchschnittliche Blockzeit von etwa 12 Sekunden [2]. Die Blockgröße wird durch das Gas-Limit pro Block begrenzt, was die Transaktionskapazität direkt beeinflusst. Der bisher höchste dokumentierte Wert für Transaktionen pro Tag (TPD) auf der Ethereum-Blockchain lag bei 1.961.144, was etwa 22,70 Transaktionen pro Sekunde (TPS) entspricht [3].

`Konsensmechanismus` Seit dem Merge im September 2022 verwendet Ethereum den Proof-of-Stake (PoS) Konsensmechanismus. PoS ist energieeffizienter und ermöglicht schnellere Blockzeiten sowie eine bessere Grundlage für zukünftige Skalierungsverbesserungen wie Sharding [4].

`Gasgebühren` Ethereum verwendet einen dynamischen Preismechanismus (EIP-1559) für Transaktionsgebühren, der die Vorhersagbarkeit verbesserte [5]. Trotz dieser Effizienzsteigerung begrenzt die limitierte Transaktionskapazität die Skalierbarkeit erheblich. Bei hoher Netzwerkauslastung führt dies zu stark ansteigenden Gebühren, was die Interaktion mit der Blockchain verteuert und die Zugänglichkeit einschränkt.

3.2 Derzeitiger Entwicklungsstand und Roadmap

Der Übergang von Ethereum zu seiner neuen Architektur (früher als Ethereum 2.0 bezeichnet)[1] zielt darauf ab, die Skalierbarkeit, Sicherheit und Nachhaltigkeit des Netzwerks zu verbessern. Die wichtigste Veränderung ist der Wechsel vom Proof-of-Work (PoW) zum PoS Konsensmechanismus, der mit der Einführung der Beacon Chain im Dezember 2020 begonnen hat.

Die Roadmap umfasst mehrere Phasen:

1. Phase 0 (abgeschlossen): Die Beacon Chain wurde am 1. Dezember 2020 eingeführt. Sie dient als Rückgrat für die neue Ethereum-Architektur, indem sie die Validator-Knoten koordiniert.
2. The Merge (abgeschlossen): Am 15. September 2022 wurde die bestehende Ethereum-Blockchain (Ausführungsschicht) erfolgreich mit der Beacon Chain (Konsensschicht) fusioniert. Dies führte zu einer drastischen Reduzierung des Energieverbrauchs um etwa 99,95% [4].
3. Sharding (in Entwicklung): Die Einführung von Shard Chains zur Verbesserung der Skalierbarkeit ist noch in Arbeit. Sharding wird die Ethereum-Blockchain in mehrere parallele Ketten (Shards) aufteilen, was die Transaktionskapazität erheblich erhöhen soll. Die genaue Implementierung und der Zeitplan wurden angepasst und sind noch in der Entwicklung. Ein Schritt, der dafür bereits implementiert wurde:

    - EIP-4844 (Proto-Danksharding): Dieses Upgrade führt eine neue Art von Transaktionen mit “Blobs” ein, um die Datenverarbeitung zu optimieren. Es ist ein Zwischenschritt zum vollständigen Danksharding, das die endgültige Form des Sharding in Ethereum darstellen wird [7].

Diese Veränderungen, insbesondere die Shard Chains, sollen Ethereum langfristig skalierbar machen. Die vollständige Umsetzung dieser Roadmap wird allerdings voraussichtlich noch mehrere Jahre in Anspruch nehmen, wobei kontinuierliche Verbesserungen und Anpassungen zu erwarten sind.
Parallel zu diesen Entwicklungen auf der Main-Chain (L1) haben sich verschiedene Layer-2-Skalierungslösungen (L2) etabliert.

[1] Der Begriff Ethereum 2.0 wird offiziell nicht mehr vom Entwicklerteam verwendet. Stattdessen spricht man von der Konsensschicht (ehemals Ethereum 2.0) und der Ausführungsschicht (ehemals Ethereum 1.0) [6].

3.3 Derzeitige Skalierungslösungen

Aufgrund der anhaltenden Limitationen der Ethereum-Main-Chain haben sich verschiedene Skalierungslösungen etabliert, um die Netzwerkkapazität zu erhöhen und Transaktionskosten zu senken. Die am häufigsten verwendeten und vielversprechendsten Lösungen sind:

`Rollups` Diese L2-Lösungen arbeiten außerhalb der Ethereum-Blockchain (off-chain), um Transaktionen schneller und kostengünstiger abzuwickeln [8].

`Sidechains` Eigenständige Blockchains, die parallel zu Ethereum laufen [9].

`Validium` Eine neuere Variante von Zero-Kowledge Rollups mit off-chain Datenspeicherung für noch höhere Skalierbarkeit [10].

Weitere Lösungen, die in der Vergangenheit relevant waren oder sich in der Entwicklung befinden:

`State Channels` Ermöglichen schnelle off-chain Transaktionen zwischen Teilnehmern. Weniger verbreitet als Rollups [11].

`Plasma` Eine frühere L2-Lösung, die an Bedeutung verloren hat, aber in einigen Projekten noch verwendet wird [12].

`Volitions` Eine Hybridlösung in Entwicklung, die Elemente von Rollups und Validium kombiniert [13].

Rollups haben sich als führende Skalierungslösung etabliert [8, p. 304]. Im folgenden Kapitel wird detailliert auf Rollups eingegangen, um ihre Funktionsweise, die verschiedenen Arten und ihre Bedeutung für die Zukunft der Blockchain-Skalierung zu beleuchten.

4 Rollups

Für Rollups ist es entscheidend, die Echtheit von Transaktionen und den korrekten Zustand des Systems zu beweisen, um die Sicherheit und Integrität des Netzwerks zu gewährleisten. Beweismechanismen lösen das Vertrauensproblem, das durch die Auslagerung der Transaktionsverarbeitung von L1 auf L2 entsteht.
Ein Beispiel verdeutlicht die Herausforderung: Wenn Alice in einer L2-Transaktion 10 ETH an Bob überweist, ist diese Berechnung für die Ethereum-Blockchain nicht direkt sichtbar, es sei denn, sie wird explizit an L1 übermittelt. Ohne robuste Prüfmechanismen könnte ein unehrlicher L2-Sequencer – der für die Weiterleitung der Transaktionen und Zustandsinformationen von L2 an L1 verantwortlich ist – die Transaktionsdetails manipulieren. So könnte er Bob nur 9 ETH überweisen, 1 ETH für sich behalten und anschließend gefälschte Zustandswurzeln an L1 übermitteln, um den Betrug zu verschleiern.
Um solche Risiken zu vermeiden, setzen Rollups auf zwei unterschiedliche Beweismechanismen: Optimistic und Zero-Knowledge (ZK) Rollups. Optimistic Rollups verwenden Challenge-Perioden und Anreizsysteme, die es den Netzwerkteilnehmern ermöglichen, verdächtige Transaktionen innerhalb eines festgelegten Zeitraums anzufechten. Dadurch werden betrügerische Aktivitäten erkannt und verhindert.
ZK-Rollups hingegen beweisen die Gültigkeit von Transaktionen kryptografisch mithilfe von Zero-Knowledge-Proofs. Diese garantieren die Integrität, ohne dabei die Details der Transaktionen offenzulegen. Beide Ansätze verfolgen das Ziel, Vertrauen in die Skalierungslösungen zu schaffen, unterscheiden sich jedoch grundlegend in ihrer Herangehensweise.

Im Folgenden werden diese beiden Varianten im Detail betrachtet.

4.1 Optimistic Rollups

Optimistic Rollups sind eine Skalierungslösung, bei der eine optimistische Annahme getroffen wird, dass alle Transaktionen korrekt sind. Nur bei Verdacht auf Betrug wird eine Überprüfung durchgeführt.

4.1.1 Funktionsweise

1. Transaktionsbündelung: Optimistic Rollups bündeln eine Vielzahl von Transaktionen in einem Rollup-Block und veröffentlichen nur eine Zusammenfassung dieses Blocks auf der Ethereum-Main-Chain. Dies reduziert die Datenmenge auf L1 und senkt somit die Gasgebühren erheblich.

2. Optimistische Annahme: Transaktionen werden ohne sofortige Überprüfung auf Korrektheit verarbeitet. Es wird davon ausgegangen, dass sie korrekt sind, daher der Name *optimistisch*.
3. Fraud Proofs: Es gibt eine Frist (typischerweise 7 Tage), innerhalb derer jeder Teilnehmer im Netzwerk eine Betrugsüberprüfung (Fraud Proof) initiieren kann, wenn eine falsche Transaktion vermutet wird. Wird ein Betrug nachgewiesen, wird die fehlerhafte Transaktion rückgängig gemacht und der betrügerische Akteur bestraft.
4. Sicherheit: Die Sicherheit von Optimistic Rollups beruht auf der Fähigkeit des Netzwerks, Betrug zu erkennen und zu bestrafen. Diese Methode ermöglicht eine hohe Skalierbarkeit und Reduzierung der Kosten, da nur wenige Transaktionen überprüft werden. Die Finalität wird auf L1 festgehalten, was bedeutet, dass die endgültigen Zustandsänderungen auf der Ethereum-Main-Chain verankert sind [14].

4.1.2 Infrastruktur

Die Infrastruktur eines Optimistic Rollups ist typischerweise wie folgt aufgebaut:

- **Sequencer**: Ein zentraler Sequencer bündelt Transaktionen und erstellt neue Blöcke auf L2.
- **Smart Contracts auf Ethereum L1**: Mehrere Smart Contracts steuern und sichern das Rollup-System, darunter z.B.:
  - ein Contract zur Speicherung der Transaktionsdaten und Zustandsübergänge (Zustandswürzeln) von L2
  - ein Bridge Contract für Token-Transfers zwischen L1 und L2
  - ein Contract zur Verarbeitung von Fraud Proofs [15]
- **Validatoren**: Jeder Netzwerkteilnehmer kann als Validator agieren, indem er L2 Transaktionen beobachtet und bei Entdeckung eines Fehlers Fraud Proofs einreicht.
- **Finalisierung**: Transaktionen und der Rollup-Zustand gelten als finalisiert, wenn innerhalb der Challenge-Periode kein erfolgreicher Fraud Proof eingereicht wird [16].

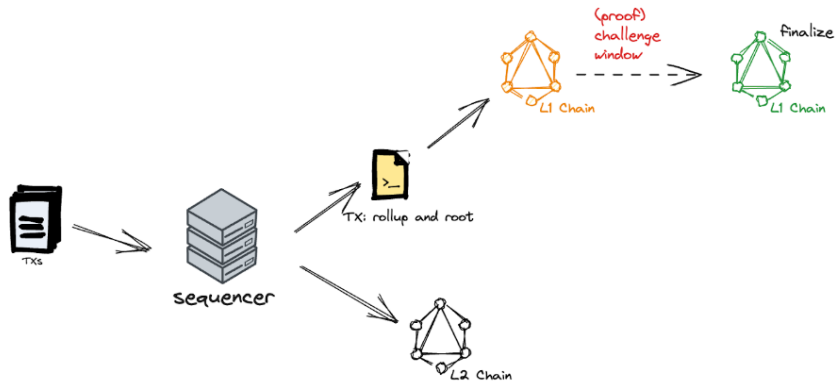


Abbildung 4.1: Optimistic Rollup Struktur
Quelle [17]

4.1.3 Fraud Proof Ablauf

Ein Fraud Proof läuft typischerweise folgendermaßen ab:

1. Verdacht auf fehlerhafte Transaktion: Ein Beobachter identifiziert eine möglicherweise fehlerhafte Transaktion oder einen ungültigen Zustandsübergang im L2.
2. Einleitung des Fraud Proofs: Der Beobachter initiiert den Fraud Proof, indem er eine Transaktion auf L1 an einen spezifischen Smart Contract sendet. Dabei muss er eine Kaution hinterlegen, um böswillige Anfechtungen zu verhindern, und Beweise für seine Behauptung vorlegen.
3. Ausführung des Fraud Proofs: Der Fraud Proof wird on-chain auf L1 durch den zuständigen Smart Contract durchgeführt.
4. Wiederholung der Transaktion: Der Smart Contract rekonstruiert den Zustand des Rollups unmittelbar vor der strittigen Transaktion und führt diese und die damit verbundenen Zustandsübergänge erneut in einer virtuellen Maschine (VM) innerhalb des Smart Contracts aus.
5. Vergleich der Ergebnisse: Der Smart Contract vergleicht das Ergebnis seiner Ausführung mit dem vom Rollup behaupteten Ergebnis.
6. Entscheidung: Wenn der Smart Contract feststellt, dass das Ergebnis seiner Ausführung von dem behaupteten Ergebnis abweicht, wird der Fraud Proof als erfolgreich betrachtet. Andernfalls wird er abgelehnt.
7. Konsequenzen: Bei einem erfolgreichen Fraud Proof wird der fehlerhafte Zustand rückgängig gemacht und der Rollup kehrt zu einem früheren, korrekten Zustand zurück. Der Beobachter, der den Fraud Proof eingeleitet hat, erhält in der Regel eine Belohnung, während der Sequenzer und möglicherweise andere beteiligte Parteien, abhängig von der spezifischen Implementierung, bestraft werden.
8. Finalisierung: Wenn kein erfolgreicher Fraud Proof innerhalb der Challenge-Periode eingereicht wird, gilt der Zustand des Rollups als endgültig [18].

4.1.4 Zentralisierungsaspekt

Derzeitige Optimistic Rollups haben zentrale Komponenten. Die Sequenzierung der Transaktionen wird von einem zentralen Operator/Sequencer durchgeführt, was bedeutet, dass dieser Operator die Reihenfolge der Transaktionen beeinflussen, Transaktionen zensieren sowie einen Single Point of Failure darstellen kann [14].

4.1.5 Vor- und Nachteile

Vorteile von Optimistic Rollups:

- EVM-Kompatibilität: Vollständige Kompatibilität mit der Ethereum Virtual Machine, was die Portierung bestehender Smart Contracts erleichtert [19].
- Skalierbarkeit: Erhöhte Transaktionsgeschwindigkeit und reduzierte Kosten im Vergleich zu L1.
- Sicherheit: Nutzen die Sicherheit von Ethereum durch die Veröffentlichung von Transaktionsdaten auf L1.

Nachteile von Optimistic Rollups:

- Verzögerung bei der Auszahlung: Aufgrund der Challenge-Periode kann es zu Verzögerungen bei der Übertragung von Geldern von L2 auf L1 kommen [20].
- Vertrauensannahme: Es wird ein gewisses Maß an Vertrauen in die Ehrlichkeit der Mehrheit der Teilnehmer vorausgesetzt.
- Kosten für Fraud Proofs: Fraud Proofs sind aufgrund der notwendigen On-Chain-Berechnungen kostspielig.
- Komplexität der Fraud Proofs: Die Implementierung und Verwaltung von Fraud Proofs kann komplex sein.

4.1.6 Fazit

Optimistic Rollups, insbesondere durch Projekte wie Optimism [21] und Arbitrum [22], spielen gegenwärtig eine wichtige Rolle bei der Skalierung von Ethereum. Sie ermöglichen Entwicklern, effiziente und kostengünstige dApps zu erstellen und haben maßgeblich zur Verbesserung der Transaktionskapazität und Kosteneffizienz des Ethereum-Ökosystems beigetragen. Als aktuelle Lösung für Skalierungsherausforderungen bleiben sie relevant und weitverbreitet.
Dennoch weisen Optimistic Rollups einige inhärente Schwachstellen auf, die ihre langfristige Stellung in Frage stellen. Die längere Challenge-Periode und die Komplexität bei der Erstellung von Fraud Proofs stellen fortlaufende Herausforderungen dar. Zudem wirft die aktuelle Struktur der Optimistic Rollups Fragen zur nachhaltigen Dezentralität

und Sicherheit auf. Derzeit verlassen sich die meisten Implementierungen auf einen zentralen Sequenzer, was zwar Vertrauen in einzelne Institutionen wie die Optimism Foundation erfordert, aber die Notwendigkeit von Fraud Proofs minimiert.
Die geplante Dezentralisierung der Sequenzierung bringt dabei neue Herausforderungen mit sich. Das optimistische Modell, das von der Ehrlichkeit der Teilnehmer ausgeht, könnte in einem vollständig dezentralen Netzwerk an seine Grenzen stoßen. Spieltheoretisch betrachtet besteht die Gefahr, dass Teilnehmer in einem dezentralen System versuchen werden, das Netzwerk zu ihrem eigenen Vorteil auszunutzen. Dies könnte die Effektivität des Fraud-Proof-Systems auf die Probe stellen und neue Sicherheitsrisiken mit sich bringen.
Vitalik Buterin, der Mitgründer und Ideengeber von Ethereum, hat in seinem Artikel “The different types of ZK-EVMs” die Ansicht geäußert, dass die Zukunft eher in ZK-Proof-Systemen liegt [23]. Diese Einschätzung deutet darauf hin, dass, obwohl Optimistic Rollups gegenwärtig eine wichtige Rolle spielen, andere Skalierungslösungen, insbesondere solche basierend auf Zero-Knowledge-Kryptographie, zukünftig an Bedeutung gewinnen werden.
Während Optimistic Rollups also weiterhin eine wichtige Option für Entwickler darstellen, deuten die technologischen Trends und die inhärenten Herausforderungen darauf hin, dass ihre Verwendung in Zukunft möglicherweise zugunsten fortschrittlicherer Lösungen abnehmen könnte.

4.2 Zero-Knowledge (ZK) Rollups

ZK-Rollups nutzen Zero-Knowledge-Kryptographie, um Beweise zu erstellen, die Transaktionen verifizieren und ihre Korrektheit sicherstellen, ohne dabei die eigentlichen Daten zu veröffentlichen. Das Loopring Protokoll war der erste ZK-Rollup auf Ethereum und wird in diesem Kapitel und der fortlaufenden Arbeit als praktisches Beispiel dieser Technologie betrachtet.

4.2.1 Loopring

Loopring ist 2017 von dem Softwareingenieur und Unternehmer Daniel Wang gegründet worden. Mit Hauptsitz in Shanghai, China, positioniert sich Loopring als L2-Protokoll auf Ethereum und zielt darauf ab, die Skalierbarkeit und Effizienz dezentraler Kryptobörsen (DEXs) zu verbessern [8, p. 310]. Das Projekt startete mit einem Initial Coin Offering (ICO) im August 2017, bei dem 45 Millionen US-Dollar eingesammelt wurden. Allerdings musste ein Großteil dieser Mittel aufgrund verschärfter Regulierungen in China zurückgegeben werden [24]. Trotz dieser anfänglichen Herausforderung entwickelte das Team das Protokoll weiter und führte ca. im Dezember 2020 Loopring v3.6 ein, was die Einführung von NFT-Operationen mit sich brachte. Loopring nutzt innovative Technologien wie ZK-Kryptografie und einen Orderring-Mechanismus, um die Leistungsfähigkeit und

Liquidität von DEXs zu verbessern [25, 26].

4.2.2 Funktionsweise

1. Transaktionsbündelung: Ähnlich wie bei Optimistic Rollups werden Transaktionen in einem Rollup-Block gebündelt und eine Zusammenfassung auf der Ethereum-Main-Chain veröffentlicht.
2. Zero-Knowledge-Proofs: ZK-Rollups erzeugen einen kryptografischen Beweis, um die Korrektheit der gebündelten Transaktionen zu verifizieren. Dieser Beweis wird zusammen mit der Zusammenfassung des Rollup-Blocks auf Layer 1 veröffentlicht.
3. Verifizierung: Die ZK-Proofs ermöglichen der Ethereum Blockchain, die Korrektheit der Transaktionen über die Smart Contracts des Protokolls zu überprüfen, ohne die tatsächlichen Transaktionsdaten zu kennen. Dies gewährleistet sowohl Datenschutz als auch Sicherheit.
4. Sicherheit: Die Korrektheit jeder Transaktion und jedes Rollup-Blocks wird durch einen ZK-Proof verifiziert. Auch bei ZK-Rollups wird die Finalität auf Layer 1 festgehalten.

4.2.3 Infrastruktur

Die Infrastruktur des Loopring Protokolls ist wie folgt aufgebaut:

- **Relayer**: Der Relayer (Operator) ist ein zentraler Bestandteil der Infrastruktur, der als Koordinator fungiert. Er empfängt Transaktionen von Nutzern, aggregiert sie zu Rollup-Blöcken und erstellt ZK-Proofs, die die Gültigkeit der Transaktionen und Blöcke bestätigen [26, Loopring Relayer].
- **Smart Contracts auf Ethereum L1**: Loopring nutzt ein System von Smart Contracts auf der Ethereum-Main-Chain, die eine zentrale Rolle in der Infrastruktur spielen. Diese Smart Contracts erfüllen mehrere wichtige, aber auch sehr komplexe Funktionen innerhalb des Ökosystems [27]. Essentielle Beispiele sind:

  – Verwaltung von Nutzerkonten: Ein Smart Contract ermöglicht die Einzahlung von Guthaben und der Registrierung von Nutzern auf Layer 2 [28].
  – Verifizierung von ZK-Proofs: Ein Smart Contract überprüft die Gültigkeit der aggregierten Transaktionsdaten und validiert den ZK-Proof vor der Aufnahme in der Blockchain [29].

  Die Smart Contracts von Loopring wurden mehrmals auditiert, der aktuellste Audit ist von der Loopring Version 3.6 durchgeführt von least Authority [30].

- **Finalisierung**: Die Finalität der Transaktionen wird auf Layer 1 festgehalten. Sobald die Transaktionen in einem Rollup-Block auf Layer 1 veröffentlicht sind, können sie nicht mehr rückgängig gemacht oder geändert werden [26, 31].

4.2.4 ZK-Proofs und Trusted Setup

Zero-Knowledge-Proofs bilden das Herzstück der Loopring-Technologie. Loopring verwendet *Zero-Knowledge Succinct Non-Interactive Argument of Knowledge* (zkSNARK)-Beweise [32]. Im Gegensatz zu Optimistic Rollups erfordern die ZK-Rollups, die zkSNARKs verwenden, ein Trusted Setup [8, p. 309].
Ein Trusted Setup ist ein kryptografischer Prozess, bei dem entscheidende Parameter (zufällige Zahlen) für die Sicherheit und Funktionalität des Protokolls generiert werden. Dieser Vorgang beinhaltet die Erzeugung vertraulicher Informationen, die nach der Generierung der öffentlichen Parameter vernichtet werden müssen. Die Sicherheit des gesamten kryptografischen Systems hängt davon ab, dass diese Geheimnisse nicht kompromittiert werden. Um das Vertrauen in diesen Prozess zu stärken, wird er oft als *Multi-Party-Computation* durchgeführt, bei der mehrere unabhängige Parteien beteiligt sind. Solange mindestens eine dieser Parteien ehrlich handelt und ihre Geheimnisse vernichtet, bleibt das System sicher. Die Sicherheit und Vertrauenswürdigkeit dieses Setups ist kritisch, da eine Kompromittierung die Integrität des gesamten Systems gefährden könnte [33].

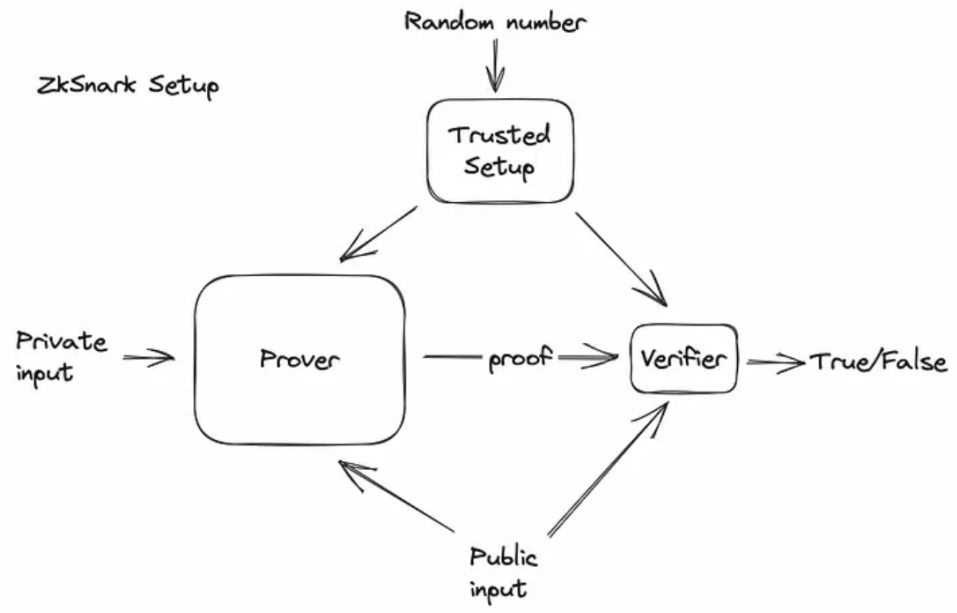


Abbildung 4.2: zkSNARK Setup mit Trusted Setup
Quelle [34]

4.2.5 Zentralisierungsaspekt

Loopring weist in seiner aktuellen Implementierung zentralisierte Elemente auf, insbesondere durch den Einsatz eines zentralen Relayers. Trotz dieser Zentralisierung nutzt Loopring eine Kombination aus fortschrittlichen kryptografischen Techniken und Smart Contracts auf der Ethereum-Blockchain, um die Sicherheit und Integrität des Systems zu gewährleisten. Alle Operationen werden durch ZK-Proofs abgesichert und auf der Ethereum-Main-Chain verifiziert. Nutzer müssen primär auf die korrekte Ausführung der Smart Contracts auf Ethereum vertrauen, da alle Transaktionen und Zustandsänderungen durch diese Beweise validiert werden. Allerdings muss auch auf die korrekte Ausführung des Trusted Setups vertraut werden, da dies die Grundlage für die Sicherheit und Integrität der ZK-Proofs bildet. Somit bilden das Trusted Setup und die zentrale Rolle des Relayers potenzielle Single Points of Failure, insbesondere der Relayer hinsichtlich der Systemverfügbarkeit [35].

4.2.6 Governance und LRC-Token

Das Loopring-Whitepaper beschreibt das Konzept eines sozialen Protokolls, das auf Koordination zwischen Mitgliedern basiert, um effektiv auf gemeinsame Ziele hinzuarbeiten [36, chap. 8.2]. Aus diesem Konzept ist der LRC-Token entstanden, der heute eine zentrale Rolle im Loopring-Ökosystem spielt.

LRC, ein ERC-20-Token, erfüllt mehrere wichtige Funktionen im Loopring-Protokoll:

1. Funding des Protokolls: LRC wurde verwendet, um die Entwicklung und Wartung des Loopring-Protokolls zu finanzieren.
2. Gebühren und Belohnungen: Transaktionsgebühren können in LRC bezahlt und an verschiedene Teilnehmer verteilt werden.
3. Staking: LRC-Inhaber können ihre Token staken, um einen Anteil an den Protokollgebühren zu verdienen [37]. Das Staking erfordert eine Mindestdauer von 90 Tagen, und die Belohnungen stammen aus einem Teil der Protokollgebühren [38].
4. Sicherheit: DEX-Betreiber müssen LRC staken, um als Sicherheit zu dienen und Protokollgebühren zu reduzieren [39].

Die Governance des Loopring-Protokolls wird durch die Loopring DAO unterstützt, die in Q3 2021 eingeführt wurde [40]. Diese DAO ermöglicht es LRC-Token-Besitzern, über wichtige Parameter und Entscheidungen abzustimmen. Zu den bereits getroffenen Entscheidungen gehört ein Voting über Staking-Mechanismen, was zeigt, dass die DAO bereits aktiv war [41].

Geplante Funktionen der DAO:

- Stimmrechte für Token-Inhaber: Die DAO gewährt LRC-Token-Besitzern Stimmrechte zur Abstimmung über Protokollparameter.
- Verteilung von Protokollgebühren: Die DAO soll über die Verteilung von 10% aller Protokollgebühren entscheiden. Diese können für Rückkäufe von LRC zum Verbrennen, Zuschüsse oder weitere Anreize verwendet werden [40].
- Steuerung von Smart Contract Updates: Die DAO soll zukünftig Einfluss auf Smart Contract Updates haben, um Kontinuität und Sicherheit zu gewährleisten.

Obwohl die Loopring DAO bereits erste Schritte unternommen hat, sind weitergehende Entwicklungen derzeit nicht auf der Roadmap priorisiert, wie im Community Call von Januar 2024 von Byron (einem Loopring Mitarbeiter) erwähnt wurde [42]. Der Fokus liegt aktuell auf anderen Features zur Förderung der Adaption des Ökosystems. Dennoch bleibt die Weiterentwicklung der DAO ein Thema für die Zukunft.

4.2.7 Vor- und Nachteile

Vorteile von ZK-Rollups:

- Sofortige Entgültigkeit: Da die Transaktionen durch ZK-Proofs verifiziert werden, gibt es keine Verzögerung bei der Auszahlung.
- Hohe Sicherheit: Die kryptografischen Beweise bieten eine hohe Sicherheit und verhindern Betrug effektiv.
- Datenschutz: ZK-Rollups können Transaktionen verifizieren, ohne sensible Daten preiszugeben.

Nachteile von ZK-Rollups:

- Rechenaufwand: Die Erstellung von ZK-Proofs ist sehr rechenintensiv.
- Komplexität der Implementierung: Die Implementierung von ZK-Rollups ist komplex und erfordert spezielle Kenntnisse in Kryptografie.
- Eingeschränkte EVM-Kompatibilität: Manche ZK-Rollups, einschließlich Loopring, haben Einschränkungen in der Kompatibilität mit der EVM, was die Portierung bestehender Smart Contracts erschweren oder sogar unmöglich machen kann.

4.2.8 Fazit

Loopring implementiert die ZK-Rollup-Technologie auf innovative Weise, unterscheidet sich jedoch in seiner Architektur von anderen L2-Lösungen. Im Kern ist Loopring eher als eine dezentralisierte Anwendung (dApp) zu betrachten, die auf Ethereum aufbaut, anstatt eine vollständig offene L2-Plattform zu sein.
Ein entscheidender Aspekt von Loopring ist, dass es keine offene EVM anbietet. Nutzer können keine eigenen Smart Contracts auf Loopring veröffentlichen, sondern sind

an die vordefinierten Funktionalitäten gebunden. Das Protokoll stellt eine vorgefertigte Infrastruktur mit spezifischen API-Endpunkten zur Verfügung. Entwickler und Nutzer interagieren mit dem Protokoll über diese APIs, die an den zentralen Relayer gesendet werden. Diese Struktur ermöglicht es Loopring, eine hochgradig optimierte und skalierbare Lösung anzubieten, beschränkt jedoch die Flexibilität für komplexere oder maßgeschneiderte Smart-Contract-Funktionalitäten [43]. Diese Einschränkung ist eine bewusste Designentscheidung, die die Effizienz und Sicherheit des Systems erhöht, indem nur sorgfältig geprüfte und optimierte Funktionen zugelassen werden.

5 Taiko: Eine hybride Layer-2-Lösung

Taiko ist ein neuer Rollup für Ethereum, der einen einzigartigen Ansatz verfolgt, indem er Elemente von Optimistic und ZK Rollups kombiniert. Diese Hybridlösung zielt darauf ab, die Vorteile beider Technologien zu nutzen und gleichzeitig deren Nachteile zu minimieren. Taiko ist seit Mai 2024 auf dem Ethereum-Mainnet aktiv. Der erste Block wurde am 25.05.2024 erstellt [44].

Taiko wurde im Jahr 2022 von Daniel Wang, dem Gründer von Loopring, und Brecht Devos ins Leben gerufen [45]. Taiko zeichnet sich durch vollständige Ethereum-Äquivalenz (Type 1 zkEVM) aus. Somit ist Taiko in der Lage, jede Ethereum-Anwendung ohne Modifikationen auszuführen. Besonders hervorzuheben ist Taikos Implementierung als “Based Rollup” [46]. Das bedeutet, dass die Transaktionssequenzierung auf Taiko direkt von Ethereum-L1-Validatoren durchgeführt wird. Dadurch erbt Taiko die Lebendigkeit und glaubwürdige Neutralität von Ethereum, was zu einer maximalen Vereinfachung und erhöhten Sicherheit führen soll. Was Taiko weiter auszeichnet, ist sein Ansatz zur Dezentralisierung. Als eines der ersten L2-Projekte startet Taiko mit einem vollständig dezentralisierten Ansatz mit einem zukünftigen Netzwerk aus Proposern und Provern. Diese Rollen soll jeder Teilnehmer in der Zukunft des Netzwerks ohne Einschränkungen übernehmen können. Ein weiteres Hauptmerkmal von Taiko ist das Konzept der “Inception Layers”, das es ermöglicht, Taiko nicht nur als L2 auf Ethereum, sondern auch als Layer 3 auf bestehenden L2-Lösungen zu implementieren. Das Protokoll verfolgt das Ziel, die Skalierbarkeit von Ethereum erheblich zu verbessern, während es gleichzeitig die Prinzipien von Sicherheit, Dezentralisierung und Genehmigungsfreiheit (*Permissionlessness*) wahrt. Es ist allerdings anzumerken, dass viele dieser Konzepte neu sind und noch nicht ausgiebig in der Praxis getestet wurden. Das Design dieses Rollups ändert sich aktiv [47].

5.1 Dezentralisierung als Kernprinzip

Taikos Infrastruktur basiert auf einem dezentralen Ansatz, bei dem es keinen zentralen Sequenzer oder Relayer gibt. Stattdessen kann jeder Netzwerkteilnehmer eine Taiko-Node betreiben und potenziell als Validator agieren. Die Validatoren in Taikos Netzwerk übernehmen zwei zentrale Rollen:

- **Proposer**: Diese Netzwerkteilnehmer konstruieren Rollup-Blöcke aus den L2 Transaktionen der Nutzer. Sie greifen auf den Mempool des L2-Execution-Engines zu, um ausstehende Transaktionen zu erfassen, und schlagen diese Blöcke dann dem Smart-Contract auf der Ethereum-Main-Chain zur Verifizierung vor.
- **Prover**: Sie sind verantwortlich für die Erstellung und Einreichung von Gültigkeits-

beweisen (Validity Proofs) für die vorgeschlagenen Rollup-Blöcke [48].

Dieser Ansatz fördert die Robustheit und Sicherheit des Netzwerks, indem er eine breite Beteiligung ermöglicht und Single Points of Failure vermeidet. Derzeit (Stand Dezember 2024) ist allerdings nur Taiko selbst Block Proposer. Dies wird in Ihrer Dokumentation aufgeführt [49].

5.2 Multi-Proof-System

Das Multi-Proof-System von Taiko nutzt eine Kombination verschiedener Beweistypen, um die Integrität des Netzwerks zu maximieren und die Risiken zu minimieren, die mit Fehlern oder Schwachstellen in einem einzelnen System verbunden sind.

Das geplante Proof-System von Taiko umfasst folgende Hauptkomponenten:

- **ZK-Proofs**: Taiko entwickelt mit Raiko einen Multi-Prover, der verschiedene Zero-Knowledge Virtual Machine (zkVM)-Targets unterstützt [50]. Im Gegensatz zu einer einzelnen zkEVM, die speziell für die Ausführung von Ethereum-Smart-Contracts optimiert ist, ermöglicht dieser Ansatz die Integration verschiedener Proving-Systeme:

  - Sp1: Eine zkVM, die auf RISC-V-Architektur basiert und für Effizienz und Flexibilität optimiert ist [51].
  - Risc0: Ein Zero-Knowledge-Protokoll, das ebenfalls RISC-V-Architektur nutzt und sich durch seine Einfachheit und Leistungsfähigkeit auszeichnet [52].
  - Software Guard Extensions (SGX) von Intel [53]

  Diese Vielfalt bietet mehr Flexibilität und ermöglicht eine bessere Anpassung an zukünftige Entwicklungen in der ZK-Technologie.

- **Trusted Execution Environment (TEE)**: Taiko nutzt SGX, eine von Intel entwickelte TEE-Technologie. SGX führt den gleichen Code aus, der in einer zkVM ausgeführt würde, und funktioniert ähnlich wie ein leichter Ausführungsclient. Ein Prover benötigt einen SGX-fähigen Chip, um die Beweise zu generieren [54, TEE Projektüberprüfung].

- **Guardian-Proofs**: Guardians bilden eine zusätzliche Sicherheitsebene im Taiko-Netzwerk und dienen als sekundäre Verifizierungsschicht.

Durch die Verwendung verschiedener Beweissysteme und die Integration unterschiedlicher Beweistypen strebt Taiko an, ein robustes, zukunftssicheres und hochgradig dezen-

tralisiertes Rollup-System zu schaffen. Der Ansatz zielt darauf ab, die gesamte Proof-Erstellungskette zu dezentralisieren [55] [56, Multi-proofs].

5.3 Layer 1 Smart Contracts

Die Smart Contracts auf Ethereum L1 stellen eine Reihe von Funktionen bereit, die für den Betrieb und die Sicherheit des L2-Netzwerks entscheidend sind. Sie ermöglichen das Vorschlagen, Beweisen und Verifizieren von L2-Blöcken und verwalten den gesamten Lebenszyklus dieser Blöcke. Die darin implementierten Rollup-Regeln wahren die Grundprinzipien Sicherheit, Dezentralisierung und Genehmigungsfreiheit. Ein wichtiges Merkmal des Protokolls ist das dezentrale Sequencing-Modell, das es jedem Teilnehmer ermöglicht, L2-Blöcke vorzuschlagen. Darüber hinaus koordinieren die Contracts den Prozess der Blockverifizierung über verschiedene Beweisebenen und implementieren Anfechtungsmechanismen für zweifelhafte Beweise.
Die Smart Contracts verwalten auch die ökonomischen Anreize im Netzwerk, einschließlich der Hinterlegung und Rückgabe von Bonds sowie der Verteilung von Belohnungen an die Netzwerkteilnehmer. Außerdem ermöglichen sie sichere und effiziente Vermögenstransfers zwischen Ethereum L1 und Taiko L2 [57]. Die Smart Contracts wurden von OpenZeppelin auditiert [58].

5.4 Rollup Blöcke - Zustandverifizierung

Ein Rollup-Block auf Taiko durchläuft drei Zustände: proposed, proven und verified.
Zunächst wird ein Rollup-Block als **proposed** betrachtet, wenn ein Proposer ihn aus L2-Transaktionen konstruiert und zur Verifizierung an die Ethereum-Main-Chain übermittelt hat. Proposer wählen Transaktionen aus dem Taiko L2 Mempool aus und reichen diese als Blobs oder Calldata an den Taiko-L1-Contract ein. Der Proposer eines Blocks wird automatisch zum designierten SGX-Prover für diesen Block.
Der Status **proven** wird erreicht, wenn ein gültiger Beweis für den Block erstellt wurde. Der zugewiesene SGX-Prover (in diesem Fall der ursprüngliche Proposer) muss innerhalb eines bestimmten Zeitfensters (*proofWindow*) einen SGX-Beweis einreichen, um seine Kaution zurückzuerhalten [59]. Ein Prover benötigt einen SGX-fähigen Chip [60].
Ein Block ist **verified**, wenn sein Beweis auf L1 verifiziert wurde oder die Challenge-Periode von einem Tag ohne Anfechtungen abgelaufen ist. Nach der Verifizierung wird der L2-Zustand finalisiert und kann für den Brückenprozess zwischen L1 und L2 verwendet werden. Taikos Multi-Proof-System ermöglicht verschiedene Beweisarten, darunter SGX-Beweise und zukünftig ZK-Proofs. Jeder Block kann mehrere Beweise haben, wobei der erste gültige Beweis den Block als **proven** markiert. Die Challenge-Periode erlaubt es anderen Teilnehmern, einen Beweis anzufechten, indem sie eine Anfechtungskaution hinterlegen. Bei einer Anfechtung ist ein höherwertiger Beweis erforderlich, um den Streit beizulegen [61]. Um als Proposer und SGX-Prover zu fungieren, muss ein

Teilnehmer eine Liveness-Kaution von 125 TAIKO und eine zusätzliche Validity Bond von 150 TAIKO (für SGX-Beweise) hinterlegen. Dieser Mechanismus schafft Anreize für korrekte Beweiserstellung und trägt zur Sicherheit und Integrität des Netzwerks bei [60].

5.5 Sicherheitsaspekte

Taiko implementiert einen innovativen hybriden Ansatz, der Elemente von Optimistic Rollups und ZK-Rollups kombiniert. Dieser Ansatz, bezeichnet als Based Contestable Rollup (BCR), zielt darauf ab, die Vorteile beider Systeme zu nutzen und gleichzeitig ihre jeweiligen Schwachstellen zu minimieren. Dieser Ansatz ist allerdings ganz neu und noch nicht vollständig implementiert.

Guardian-Prinzip
Das Guardian-Prinzip bei Taiko bildet einen zentralen Bestandteil des mehrstufigen Proofsystems und dient als zusätzliche Sicherheitsebene in der frühen Entwicklungsphase des Protokolls. Guardians, die derzeit vom TaikoAdmin-Multisig (Taiko Labs) ausgewählt werden, haben die Befugnis, Blöcke auf höheren Proofstufen zu validieren [61]. Das System umfasst zwei Guardian-Tiers: das 1/8- und das 6/8-Tier, wobei letzteres die höchste Autoritätsstufe darstellt [61]. Diese Struktur ermöglicht einen robusten Anfechtungsmechanismus, bei dem höhere Tiers zur Konfliktlösung herangezogen werden können. Obwohl das Guardian-System bisher eine wichtige Rolle in der Sicherheitsarchitektur von Taiko spielt, ist es als Übergangslösung konzipiert. Langfristig strebt Taiko eine vollständige Dezentralisierung an, bei der die Abhängigkeit von Guardians schrittweise reduziert wird. Diese Herangehensweise erlaubt es Taiko, in der Anfangsphase ein hohes Maß an Sicherheit zu gewährleisten, während gleichzeitig an der Entwicklung fortschrittlicherer und dezentralerer Validierungsmechanismen gearbeitet wird [46].

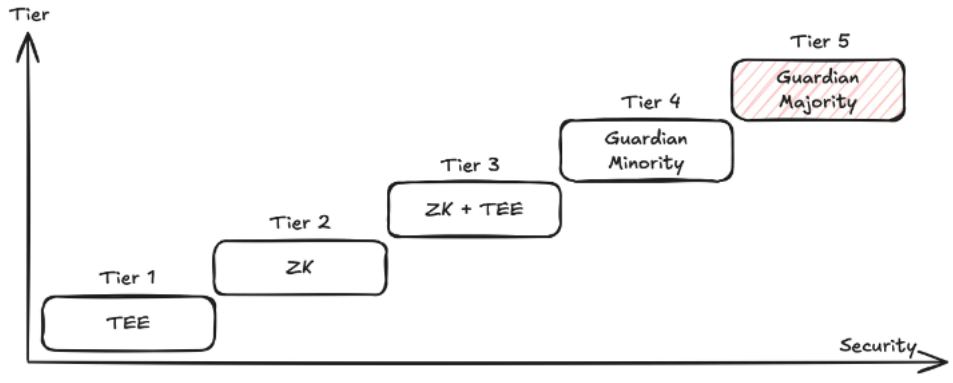


Abbildung 5.1: Taikos Multi-Proof Hirachie
Quelle [61]

Aktuelle Sicherheitsaspekte
Anfang November 2024 ähnelte Taiko in seiner Funktionsweise eher noch einem Optimistic Rollup [57]. Dies lies sich von folgenden Aspekten ableiten:

- Basiertes Sequencing: Taiko nutzt ein genehmigungsfreies Sequencing-System, bei dem jeder Teilnehmer Blöcke direkt auf dem Taiko-L1-Contract vorschlagen kann.
- Contestation-Mechanismus: Jeder SGX-Proof durchläuft eine Abkühlphase, in der Anfechtungen möglich sind. Angefochtene Blöcke erfordern einen Beweis von einer höheren Stufe.
- Fehlen eines aktiven ZK-Proof-Systems für jede Transaktion
- Vertrauensannahmen: Das System verlies sich bisher auf SGX oder Guardian Genehmigungen zur Zustandsvalidierung, was eher einem optimistischen Ansatz entsprach als einem vollständig kryptographisch gesicherten System [62, 63].

Update 11. November 2024: Taiko hat erfolgreich ZK-Proofs in sein Mainnet integriert, was einen wichtigen Meilenstein in der Entwicklung des Netzwerks darstellt. Diese Erweiterung ergänzt die bestehende Multi-Proof-Architektur, die bereits SGX-Proofs umfasst. Aktuell erfordern 3 von 100 Blöcken, die von Taiko vorgeschlagen werden, ZK-Proofs. Bis Ende 2025 plant Taiko, vollständig zu einem ZK-Rollup zu werden. Ab Januar 2025 werden ZK-Anforderungen für alle Proposer gelten, mit einem Ziel von 10-25% ZK-Abdeckung [63].

Zukünftige Sicherheitsaspekte
Taiko plant, sein Sicherheitsmodell weiter zu entwickeln:

- Hybrides Proof-System: Aktuell nutzt Taiko eine Kombination aus SGX und ZK-Proofs, was eines der umfassendsten Sicherheitssysteme im L2-Bereich etabliert.
- Schrittweise ZK-Implementierung: Derzeit erfordern 3 von 100 Blöcken ZK-Proofs, mit dem Ziel, bis Ende 2025 vollständig zu einem ZK-Rollup zu werden.
- Diversifizierung: Taiko trägt zu mehreren zkVMs bei (RISC Zero, SP1, Powdr) und nutzt verschiedene kryptographische Techniken, um die Robustheit zu erhöhen [64].

Fazit zu Sicherheitsüberlegungen
Taikos innovativer Ansatz bietet vielversprechende Sicherheitsaspekte, bringt derzeit jedoch noch einige Unsicherheiten mit sich. Die Neuartigkeit des Systems könnte unvorhergesehene Risiken bergen, da es noch nicht langfristig in der Praxis erprobt wurde. Taiko geht davon aus, dass kein Validitätsbeweis ohne jahrelange bewährte Implementierung als vollständig vertrauenswürdig gelten kann, was ihre vorsichtige Herangehensweise unterstreicht [46]. Die gegenwärtige Notwendigkeit von Guardians für fortgeschrittene Proofstufen birgt potenzielle Zentralisierungsrisiken. Es ist jedoch vorgesehen, diese Abhängigkeit mit zunehmender Systemreife schrittweise abzubauen und letztendlich

zu eliminieren. Trotz dieser Unsicherheiten zeigt Taikos hybrider Ansatz das Potential für ein robustes und anpassungsfähiges Sicherheitsmodell. Die schrittweise Integration von ZK-Proofs und die Möglichkeit dynamischer Konfigurationsanpassungen versprechen eine kontinuierliche Verbesserung der Sicherheit.

5.6 Governance und TAIKO-Token

Die Governance-Struktur von Taiko befindet sich in einem Übergangsprozess hin zu einer dezentralen Entscheidungsfindung. Die DAO ist in der Dokumentation zum Taiko-Protokoll als “in progress” gekennzeichnet [65]. Der TAIKO-Token wird eine zentrale Rolle in dieser zukünftigen Governance spielen. Aktuell werden Entscheidungen hauptsächlich von Taiko Labs getroffen.

Der TAIKO-Token erfüllt mehrere wesentliche Funktionen:

1. Governance: Mit der geplanten Einführung der Taiko DAO, werden TAIKO-Token-Besitzer Stimmrechte erhalten, um an wichtigen Entscheidungen über Protokoll-Upgrades, Gebührenstrukturen und andere kritische Aspekte des Netzwerks teilzunehmen.
2. Transaktionsgebühren: TAIKO kann zur Bezahlung von Transaktionsgebühren innerhalb des Taiko-Netzwerks verwendet werden.
3. Staking: Token-Besitzer werden die Möglichkeit haben, ihre TAIKO-Token zu staken, um zur Sicherheit des Netzwerks beizutragen und Belohnungen zu verdienen.
4. Anreize für Prover und Contester: Im Rahmen der zukünftigen Governance werden Prover, die Beweise für die Gültigkeit von Transaktionen generieren, TAIKO-Token als Validitätsbonds hinterlegen müssen. Contester, die diese Beweise anfechten, werden ebenfalls Bonds in TAIKO setzen müssen.

Taikos zukünftige Governance-Struktur und Entscheidungsfindungsprozesse sind darauf ausgelegt, eine dezentralisierte, effiziente und sichere Verwaltung des Protokolls zu gewährleisten. Das System basiert auf einem optimistischen Dual-Governance-Modell und umfasst mehrere Schlüsselkomponenten:

1. Taiko DAO:

    - TAIKO-Token-Inhaber erhalten Stimmrechte proportional zu ihrem Token-Besitz.
    - Sie können über wichtige Aspekte des Protokolls abstimmen, einschließlich:
      - Smart-Contract-Upgrades
      - Änderungen der Netzwerkparameter
      - Ressourcenallokation

- Strategische Entscheidungen für die Entwicklung des Ökosystems [65]

2. Optimistisches Token-Voting:

   - Implementiert ein System, bei dem ausgewählte Gruppen oder Mitglieder Vorschläge einreichen können.
   - Token-Inhaber haben die Möglichkeit, gegen diese Vorschläge ein Veto einzulegen.
   - Vorschläge, gegen die nach einer festgelegten Zeitspanne kein Veto eingereicht wurden, können von jedem Netzwerkteilnehmer ausgeführt werden [66].

3. Sicherheitsrat:

   - Eine Gruppe von Experten, die von der DAO gewählt wird.
   - Verantwortlich für die Handhabung von Notfallsituationen und die Gewährleistung der Protokollsicherheit.
   - Befugt, schnelle Entscheidungen in kritischen Situationen zu treffen, um das Netzwerk zu schützen [66].

4. Taiko Foundation:

   - Agiert als unterstützende Einheit für das Wachstum und die Entwicklung des Protokolls.
   - Arbeitet im Auftrag der DAO und der Token-Inhaber.
   - Fokussiert sich auf die Förderung der Adoption, Entwicklung und Verbesserung des Taiko-Ökosystems [65].

5. Smart Contract-basierte Governance:

   - Die Governance-Mechanismen sind in Smart Contracts implementiert, wie im GitHub-Repository ersichtlich. Dies gewährleistet Transparenz und Unveränderlichkeit der Governance-Regeln.

6. Graduelle Dezentralisierung:

   - Taiko plant eine schrittweise Übergabe der Kontrolle an die Community.
   - Ziel ist es, im Laufe der Zeit eine vollständig dezentralisierte Governance zu erreichen.

Dieses Governance-Modell kombiniert die Vorteile der breiten Community-Beteiligung mit der Möglichkeit, schnell auf kritische Situationen zu reagieren.

Das Gesamtangebot von 1 Milliarde TAIKO-Token wurde zum größten Teil folgendermaßen verteilt:

- 20% für Taiko Labs und das Kernteam
- 20% für die DAO-Treasury
- 16,88% für Rücklagen der Taiko Foundation
- 11,62% für Investoren
- 10% für Pioneer Airdrop
- 5% für Liquidität und Market Making
- 5% für Grants und RetroPGF
- 2% für Guardian Prover Bonds [67]

6 Vergleich der Infrastrukturen und Kommunikationskanäle

Der Fokus dieses Kapitels liegt auf den technischen Kommunikationsaspekten, insbesondere dem Relayer-System bei Loopring und den Nodes bei Taiko. Dabei wird die zugrunde liegende Infrastruktur untersucht, die den Datenaustausch und die Interaktionen mit den jeweiligen Netzwerke ermöglicht und steuert.

6.1 Looprings Infrastruktur

Im Zentrum von Looprings Infrastruktur steht der Relayer, der als Hauptschnittstelle zwischen den Nutzern und der Ethereum-Main-Chain fungiert. Der Relayer übernimmt eine Vielzahl kritischer Aufgaben, die für eine reibungslose Funktion des Loopring-Protokolls essentiell sind. Als zentraler Ansprechpunkt für Looprings Rollup-Lösung interagiert der Relayer mit jedem Nutzer und orchestriert komplexe Prozesse, die für die Effizienz und Sicherheit des Systems erforderlich sind.

Die wichtigsten Aufgaben des Relayers sind:

- Verarbeitung von Transaktionen:
  Der Relayer empfängt Transaktionsanfragen von Nutzern.
- Verwaltung des dezentralen Orderbooks:
  Der Relayer führt ein öffentliches Orderbook, das alle offenen Aufträge speichert und verwaltet.
- Berechnung und Speicherung des Rollup-Zustands:
  Der Relayer überwacht kontinuierlich den Zustand des Systems, einschließlich der Liquidität und der verfügbaren Aufträge, um sicherzustellen, dass alle Transaktionen korrekt verarbeitet werden. Der aktuelle Systemzustand wird dabei in einem Merkle-Tree gespeichert [68].
- Erstellung von Rollup-Blöcken:
  Der Relayer erstellt Rollup-Blöcke, die mehrere Transaktionen zusammenfassen und diese in einem einzigen On-Chain-Eintrag auf Ethereum speichern.
- Generierung von ZK-Proofs:
  Eine der wichtigsten Funktionen des Relayers ist die Erstellung von ZK-Proofs.
- Kommunikation mit Smart Contracts:
  Der Relayer interagiert kontinuierlich mit den Smart Contracts auf der Ethereum-Main-Chain. [36, p. 4] [26, Loopring Relayer]

6.1.1 Betrieb eines Loopring Relayers

Theoretisch ist der Betrieb eines Relayers im Loopring-Ökosystem jedem Nutzer zugänglich, der über die notwendigen technischen Kenntnisse und Ressourcen verfügt. Das Loopring-Protokoll ist durch die Smart Contracts so konzipiert, dass es prinzipiell dezentral und offen ist. Jeder, der die technischen Voraussetzungen erfüllt, kann einen Relayer aufsetzen und betreiben. Die Vorgaben des Protokolls stellen sicher, dass die Implementierung eines Relayers den festgelegten Regeln folgt und somit eine gewisse Standardisierung gewährleistet ist [36, p. 4].
Matthew Finestone, Co-Founder von Loopring, schrieb in einem Reddit Eintrag, dass der Aufwand für den Betrieb und die Verwaltung eines Relayers in der Praxis jedoch enorm sind. Die technische Komplexität des Betriebs erfordert ein tiefgreifendes Verständnis der Loopring-Architektur sowie der Zero-Knowledge-Technologie. Diese Herausforderung wird zusätzlich dadurch verstärkt, dass der Relayer-Code von Loopring Closed Source ist. Entwickler, die einen eigenen Relayer betreiben möchten, müssen daher den Großteil des Codes selbst entwickeln, was den technischen Aufwand erheblich erhöht und spezifisches Fachwissen erfordert. Neben der Codeentwicklung sind leistungsstarke Server erforderlich, um die rechenintensiven Operationen durchzuführen, die mit der Verarbeitung von Transaktionen und der Generierung von ZK-Proofs verbunden sind. Außerdem müssen hohe Sicherheitsstandards eingehalten werden, um die Integrität des Systems zu schützen und potenzielle Angriffe abzuwehren.

Aufgrund dieser vielfältigen Herausforderungen wird nur ein Relayer aktiv betrieben, und zwar vom Loopring-Team selbst. Diese Zentralisierung des Relayer-Betriebs stellt einen Kompromiss zwischen Effizienz und Dezentralisierung dar, den Loopring eingegangen ist [69].

6.1.2 API-Kommunikation

Die API-Infrastruktur von Loopring stellt einen zentralen Aspekt der Kommunikation zum Rollup dar, der die effiziente Interaktion zwischen Nutzern, Anwendungen und dem Relayer ermöglicht. Loopring hat eine spezialisierte API-Struktur entwickelt, die sowohl eine REST-API als auch eine WebSocket-API umfasst, um verschiedene Anwendungsfälle abzudecken.

REST-API
Die REST-API von Loopring bietet Entwicklern eine umfassende Schnittstelle zur Interaktion mit dem Loopring-Protokoll. Sie umfasst eine Vielzahl von Endpunkten, die es ermöglichen, Orderbook-Abfragen durchzuführen, Transaktionen zu übermitteln, Kontostände abzufragen und NFT-bezogene Operationen auszuführen. Diese API-Struktur bildet das Rückgrat für die Integration von Loopring in verschiedene Anwendungen und Dienste [43].

Websocket-API
Für Anwendungen, die Echtzeitaktualisierungen benötigen, stellt Loopring eine WebSocket-API zur Verfügung. Diese ermöglicht es, zeitnah über Änderungen im Orderbook, Handelsausführungen oder NFT-Transaktionen informiert zu werden, was insbesondere für Handelsplattformen und Portfoliomanagement-Tools von großer Bedeutung ist [70].

API-Schlüssel und Signatur
API-Schlüssel sind für den Zugriff auf bestimmte Endpunkte erforderlich und ermöglichen eine effektive Authentifizierung und Autorisierung von Anfragen. Dazu kommt eine kryptografische Signatur, die für viele API-Aufrufe benötigt wird, um sicher zu stellen, dass die Anfragen authentisch sind [71].

Im Kern eine dApp
Loopring präsentiert sich im Kern als eine spezialisierte dezentrale Anwendung (dApp) auf Ethereum, die sich auf hochoptimierte Finanz- und NFT-Funktionen konzentriert. Anstatt eine offene Plattform für eigene Smart Contracts zu bieten, stellt Loopring eine sorgfältig verwaltete Auswahl vordefinierter Funktionen über seine API bereit. Diese umfassen die gängigsten DeFi- und NFT-Operationen wie Spot- und AMM-Handel, Ein- und Auszahlungen, Transfers, Liquiditätsmanagement sowie NFT-bezogene Aktivitäten wie Minting, Handel und Transfers. Dieser fokussierte Ansatz spiegelt Loopring's Natur als spezialisierte dApp wider, die bewusst die Flexibilität für Entwickler einschränkt, um im Gegenzug überlegene Sicherheit, Leistung und Kosteneffizienz innerhalb ihres definierten Funktionsbereichs zu bieten.

6.2 Taikos Infrastruktur

Im Gegensatz zu zentralisierten Systemen wie Loopring setzt Taiko von Grund auf eine dezentrale Architektur, die die Grundprinzipien der Blockchain-Technologie widerspiegelt. Taiko basiert auf einem Netzwerk an Nodes, die als Proposer und Prover fungieren können.

6.2.1 Taiko Full Nodes

Diese Nodes sind entscheidend für die Funktionsweise von Taiko, da sie alle kritischen Aufgaben des gesamten Systems übernehmen. Dazu gehören:

- Blockvorschläge:
  In der Rolle als Proposer überwachen die Nodes den Mempool des L2-Netzwerks auf neue Transaktionen. Sie bündeln diese Transaktionen zu Blöcken und schlagen sie dem Taiko-L1-Contract vor.
- Hosting des Mempools:

Die Nodes hosten und verwalten den Mempool des L2-Netzwerks. Sie empfangen und speichern eingehende Transaktionen, bevor diese in Blöcke aufgenommen werden.

- Beweisführung:
  In ihrer Rolle als Prover generieren die Nodes Validity Proofs für die vorgeschlagenen Blöcke. Diese Beweise bestätigen die Gültigkeit der Transaktionen und des resultierenden Zustands.
- Netzwerkkommunikation:
  Die Nodes dienen als Kommunikationsknoten im Netzwerk. Sie empfangen und verbreiten Transaktionen, Blöcke und Beweise an andere Netzwerkteilnehmer und den Taiko-L1-Contract.
- Zustandsverwaltung:
  Die Nodes verwalten den aktuellen Zustand des Netzwerks. Sie führen Transaktionen in der zkVM aus und aktualisieren den Zustand entsprechend.
- Datenbankmanagement:
  Die Nodes speichern und verwalten die Datenbank aller L2-Daten, einschließlich des vollständigen Transaktionsverlaufs und des aktuellen Zustands.
- Schnittstelle für Nutzer und Anwendungen:
  Die Nodes bieten eine Schnittstelle für Benutzer und dApps, um mit dem Taiko-Netzwerk zu interagieren, Transaktionen einzureichen und Informationen abzurufen. [48]

Taiko Nodes bestehen aus zwei Softwarekomponenten, die eng miteinander interagieren, um die Funktionalität des Netzwerks zu gewährleisten.

Software-Komponenten

Die erste Komponente ist **taiko-geth**, eine speziell angepasste Version von Go Ethereum, die als Execution Engine für das Taiko-Netzwerk fungiert. Taiko-Geth ist entscheidend für die Ethereum-Kompatibilität und übernimmt folgende Kernaufgaben: Es führt Transaktionen in der zkVM aus, verwaltet den Mempool und aktualisiert den Zustand des Netzwerks. Als zkVM-Implementierung stellt taiko-geth die JSON-RPC-API bereit, die Ethereum-Entwicklern vertraut ist. Es arbeitet eng mit dem Taiko-L1-Contract zusammen, der den Ledger der L2-Transaktionsbatches enthält, und unterstützt Prover bei der Erstellung von Beweisen für die korrekte Transaktionsausführung.
Die zweite Komponente ist **taiko-client**, der aus drei Subkomponenten besteht: *driver*, *proposer* und *prover*. Der *driver* empfängt neue Blöcke vom Taiko-L1-Contract und synchronisiert sie mit taiko-geth. Der *proposer* sammelt Transaktionen aus dem Mempool und erstellt Blockvorschläge. Der *prover* koordiniert die Generierung von Validity Proofs und arbeitet zukünftig eng mit dem ZK-Prover-System Raiko [50] zusammen. Diese Komponenten spielen eine entscheidende Rolle bei der Sicherstellung der Integrität und Konsistenz des Netzwerks.
Sowohl taiko-geth als auch taiko-client sind Teil einer sich entwickelnden Technologie. Ihre genauen Funktionen und Rollen können sich im Laufe der Zeit ändern, insbesonde-

re mit der Einführung neuer Features und weiteren Optimierungen im Netzwerk. [48]

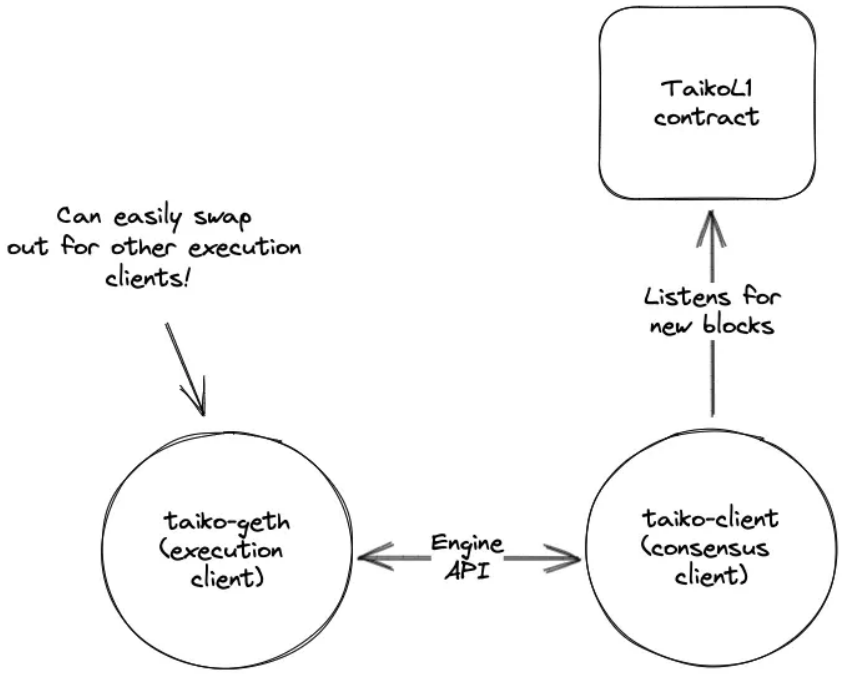


Abbildung 6.1: Taikos Software Komponenten
Quelle [48]

Verknüpfung zu Ethereum-Nodes

Alle Taiko-Nodes verbinden sich mit Ethereum-Nodes und abonnieren Ethereums Block-Events. Diese Verknüpfung ist entscheidend, da sie Taiko ermöglicht, Ethereum-Block-Events zur Synchronisation, Verifizierung und Sicherstellung einer deterministischen Blockausführung zu verfolgen. Wenn ein Taiko-Block vorgeschlagen wird, wird er zunächst einer Warteschlange im Taiko-L1-Contract auf Ethereum hinzugefügt. Nach der Verarbeitung und Validierung wird der Block dann im Taiko-L1-Contract auf Ethereum bestätigt, wodurch sein Status offiziell auf der L1-Ebene verankert wird.

Die Ankerfunktion spielt eine wichtige Rolle bei der Verknüpfung von Rollup-Blöcken mit Ethereum L1-Blöcken. Jeder Taiko-Block enthält eine deterministische `TaikoL2.anchor`-Transaktion als erste Transaktion in der vorgeschlagenen Transaktionsliste. Diese Ankertransaktion stellt sicher, dass die Rollup-Daten korrekt mit der Ethereum-Main-Chain verbunden sind und ermöglicht die Synchronisation zwischen den beiden Netzwerken [48].

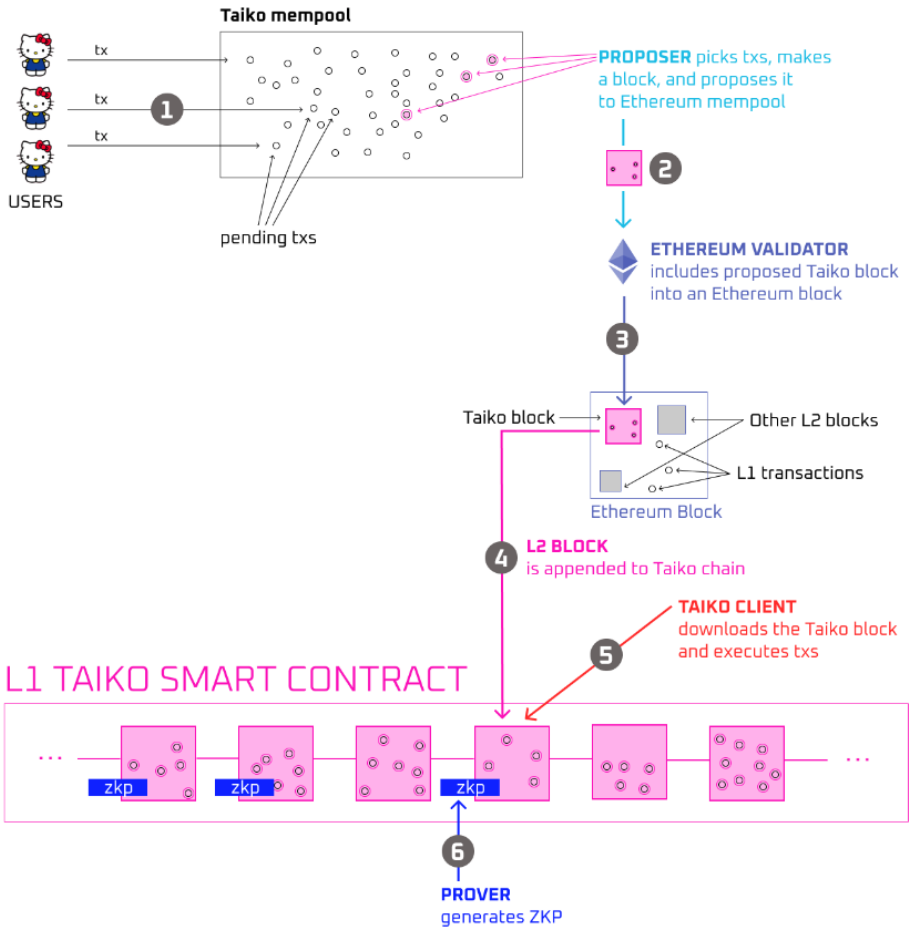


Abbildung 6.2: Taikos Ablauf und Konzept
Quelle: [72]

Eine offene zkVM

Ein wichtiger Aspekt von Taiko ist die Möglichkeit, Smart Contracts auf der zkVM veröffentlichen zu können. Diese zkVM ist vollständig kompatibel mit Ethereum, sodass bestehende Ethereum-Smart-Contracts ohne Änderungen auf Taiko eingesetzt werden können. Dies erleichtert die Migration und Integration bestehender Anwendungen in das Taiko-Ökosystem.

Permissionlessness

Die Architektur von Taiko legt großen Wert auf Dezentralisierung und Genehmigungsfreiheit (Permissionlessness). Jeder Teilnehmer im Netzwerk hat die Möglichkeit, eine Full-, Proposer- oder Prover-Node zu betreiben. Diese Offenheit trägt wesentlich zur Sicherheit und Robustheit des Netzwerks bei, da sie eine breite Verteilung der Netzwerkfunktionen ermöglicht und somit die Resistenz gegen Zensur und einzelne Ausfallpunkte erhöht.

6.2.2 Kommunikation mit Taiko Nodes

Die Kommunikation mit dem Taiko-Netzwerk erfolgt ähnlich wie bei Ethereum, wobei Nutzer Zugang zu einer Proposer-Node benötigen, um Transaktionen zu senden. Dies geschieht über Remote Procedure Call (RPC)-Endpunkte, die es ermöglichen, Ethereum-Wallets und -Tools auf Taiko-RPC-Provider umzustellen. Diese Endpunkte erlauben es den Nutzern, direkt mit dem Netzwerk zu interagieren. Dies könnte über den Betrieb einer eigenen Proposer-Node geschehen. Potenziell könnten aber auch Anbieter wie Infura RPC-Dienste für Taiko bereitstellen, was den Zugang weiter erleichtern würde, obwohl dies zum aktuellen Zeitpunkt noch nicht bestätigt ist. Derzeit bietet nur Taiko selbst einen öffentlichen RPC-Endpunkt an [73].

7 Gegenüberstellung: Sicherheit und Dezentralisierung

Kriterium	**Loopring**	**Taiko**
Sicherheit		
Konsensus-Mechanismus / Sicherheitsmodell	ZK-Rollup mit zentralem Relayer	BCR mit Multi-Tier-Proofsystem
Implementierung von Sicherheitsaudits und Best Practices	zkSNARKs; im öffentlichen Markt getestete kryptografische Techniken (*battle tested*)	SGX, Guardians, ZK(Risc0, SP1)
Open-Source-Praxis	Smart Contracts und SDK open source, Relayer Code ist closed	vollständig Open Source
Auditierung des Protokolls	least Authority (für v3.6)	OpenZeppelin
Dezentralisierung		
Verteilung der Validatoren/Nodes	zentralisierter Relayer	derzeit nur einen Proposer; zukünftig genehmigungsfrei und dezentralisiertes Proposer/Prover-Netzwerk
Governance-Struktur und Entscheidungsfindungsprozesse	DAO hat bereits über Staking-Parameter und die Verteilung von Protokoll Gebühren entschieden. Alle anderen Entscheidungen derzeit vom Loopring Team	derzeit Taiko Labs, zukünftig geplante DAO-Struktur
Abhängigkeit von zentralisierten Komponenten	starke Abhängigkeit vom zentralen Relayer sowie vom Trusted Setup	derzeit abhängig von Guardian-Struktur als höchste Autoritätsstufe; zukünftig geringere bis keine Abhängigkeit durch basiertes Sequencing und dezentrale Struktur

Tabelle 7.1: Vergleich von Loopring und Taiko in Bezug auf Sicherheit und Dezentralisierung

Bewertung

Taiko markiert einen Wendepunkt in der Bewertung von Ethereum-Rollups hinsichtlich ihrer Dezentralität. Als erster Rollup, der von Grund auf eine dezentrale Struktur anstrebt, erfordert Taiko umfangreiche Forschung und innovative Konzepte, da eine solche Umsetzung in der Praxis bisher nicht realisiert wurde.

Viele dieser vorgestellten Konzepte von Taiko sind noch theoretischer Natur und müssen erst implementiert und im Markt erprobt werden. Dennoch zeigen sich bereits vielversprechende Fortschritte: Mit der Integration eines ZK-Verifiers in ihr Mainnet beginnen

die theoretischen Konzepte konkrete Formen anzunehmen. Auch die Implementierung von SGX Proofs, die ein Beweissystem in einen optimistischen Rollup-Ansatz einführen, ist sehr fortschrittlich. Diese Entwicklungen unterstreichen das Potenzial und die Innovationskraft des Taiko-Protokolls.
Obwohl die Technologien von Taiko auf einem modernen Entwicklungsstand basieren und vielversprechend erscheinen, besteht dennoch die Möglichkeit, dass in der praktischen Umsetzung Komplikationen auftreten könnten. Im Gegensatz dazu ist Loopring bereits klare Kompromisse zugunsten der Sicherheit bei der Dezentralisierung eingegangen. Loopring ist seit längerer Zeit am Markt etabliert, und die angewandten Methoden haben sich in der Praxis bewährt und gelten als sicher.
Aktuell hat das Loopring-Protokoll im Bereich der Sicherheit einen Vorsprung, da es als vollständiger ZK-Rollup implementiert ist und für jede Transaktion einen ZK-Proof erzeugt. Dies bestätigt L2BEAT durch ihre Risiko-Aanalyse, die in 7.1 zu sehen ist. Es ist jedoch zu erwarten, dass sich dies mit bevorstehenden Updates des Taiko-Protokolls bald ändern könnte. Hinsichtlich der Dezentralisierung wird Taiko eine führende Rolle einnehmen, da die Architektur des Rollups gezielt auf Dezentralisierung ausgelegt ist. Loopring hingegen steht vor erheblichen Herausforderungen, die komplexe Arbeit des Relayers zu dezentralisieren. Allerdings zeigt das Team Bemühungen in Richtung Dezentralisierung. Loopring hat einen “Self-Help Transaction Mode” eingeführt, der es Nutzern ermöglicht, Auszahlungstransaktionen direkt on-chain (auf L1) auszuführen, falls der Relayer ausfällt. Dies deutet darauf hin, dass Loopring aktiv an Lösungen arbeitet, um die Abhängigkeit von ihrem zentralisierten Relayer zu reduzieren und Aspekte der Dezentralisierung in ihr Protokoll zu integrieren [74].

7.1 L2BEAT Risiko-Analyse

L2BEAT ist eine Analytics- und Forschungsplattform, die sich auf Ethereum L2 Skalierungslösungen konzentriert. Sie bietet Einblicke in L2-Technologien, analysiert deren Dezentralisierung und Sicherheitsannahmen und erfasst verschiedene Metriken zur Lebendigkeit und Sicherheit dieser Projekte. Die Plattform legt großen Wert darauf, Nutzern fundierte Entscheidungen durch die Bereitstellung von Sicherheitsmetriken zu ermöglichen [75].

7.1.1 Loopring

Loopring und mögliche Risiken dieses Rollups wurden von L2BEAT folgendermaßen bewertet:

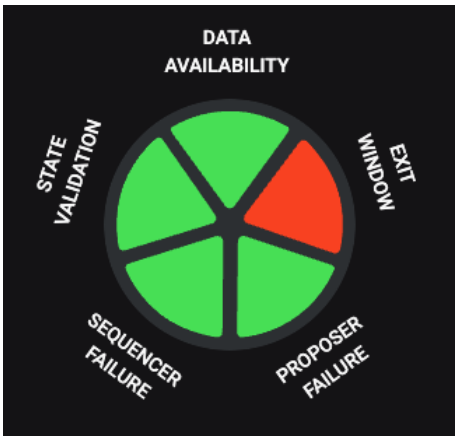


Abbildung 7.1: Risiko-Analyse Loopring - Übersicht
Quelle [76]

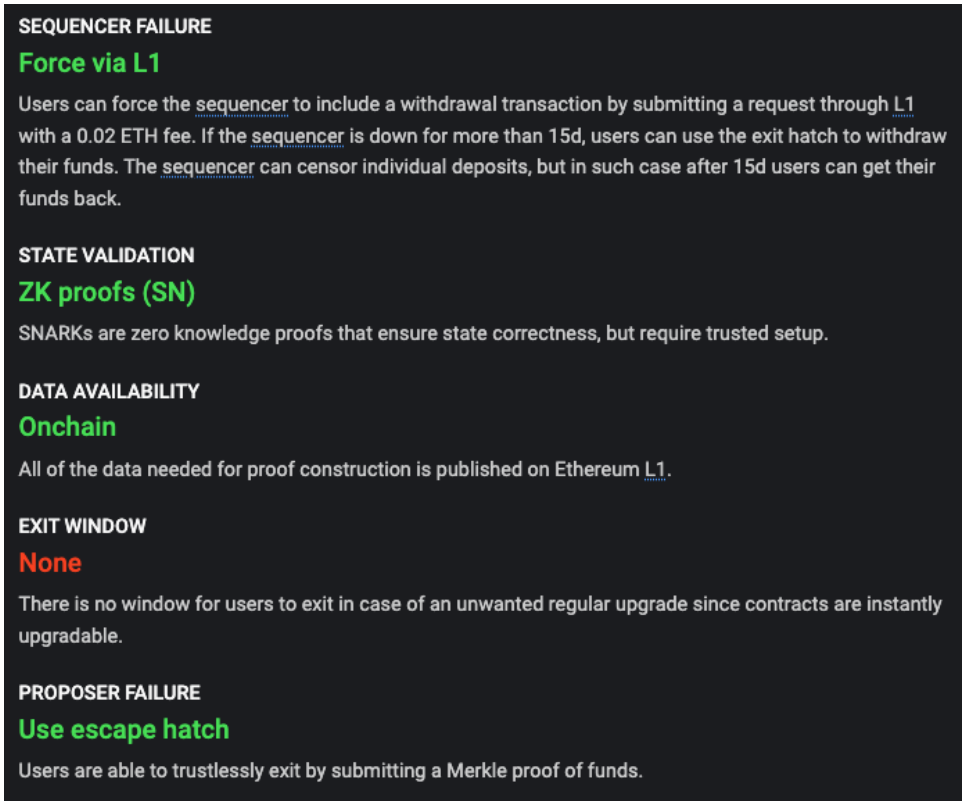


Abbildung 7.2: Risiko-Analyse Loopring - Details
Quelle [76]

7.1.2 Taiko

Taiko und mögliche Risiken dieses Rollups wurden von L2BEAT folgendermaßen bewertet:

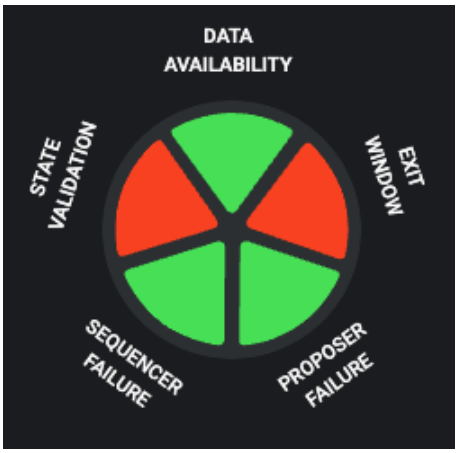


Abbildung 7.3: Risiko-Analyse Taiko - Übersicht
Quelle [77]

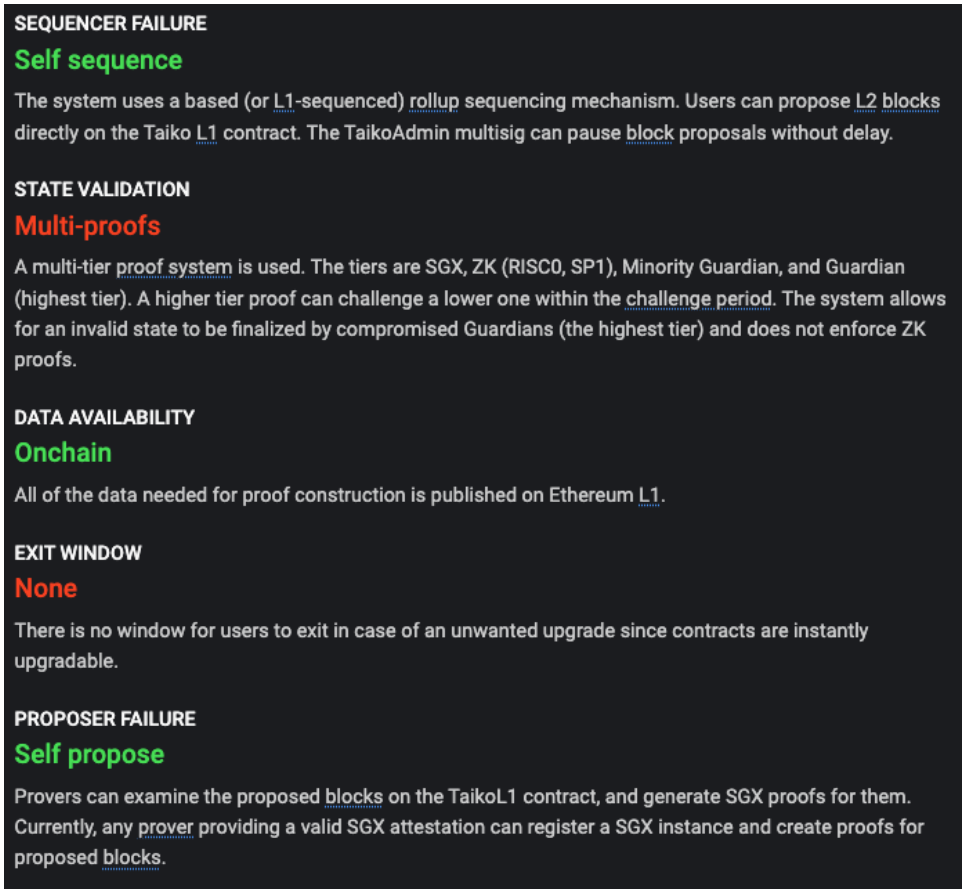


Abbildung 7.4: Risiko-Analyse Taiko - Details
Quelle [77]

8 NFT-Marktplätze: Funktionalitäten und Implementierungen

Dieses Kapitel zielt darauf ab, einen Überblick über die allgemeinen Funktionalitäten und Anforderungen von NFT-Handelsplattformen zu gewinnen.

NFT-Marktplätze bieten Nutzern die Möglichkeit, NFTs zu erstellen, zu kaufen, zu verkaufen und zu sammeln. Während einige Marktplätze erweiterte Funktionalitäten implementieren, fokussiert sich diese Arbeit auf die fundamentalen Operationen und Mechanismen, die den Kern eines NFT-Marktplatzes bilden.

8.1 Kernfunktionalitäten eines NFT-Marktplatzes

- **Minting von NFTs**
  User Story: Ein Nutzer kann digitale Vermögenswerte als NFT erstellen (*minten*).
- **Listing von NFTs**
  User-Story: NFT-Besitzer können ihre NFTs auf dem Marktplatz zum Verkauf anbieten, um potenzielle Käufer zu erreichen.
- **Kaufen von NFTs**
  User-Story: Sammler können NFTs auf dem Marktplatz durchsuchen, auswählen und erwerben, um ihre digitale Sammlung zu erweitern.

Bei der Implementierung eines NFT-Marktplatzes auf den verschiedenen Rollups müssen spezifische Aspekte berücksichtigt werden, wie z.B. die Kompatibilität mit gewissen Token-Standards und die Interaktion mit der jeweiligen L2-Infrastruktur. Da Loopring für das Minting von NFTs (und fungiblen Tokens) auf ihrem L2-Rollup nur noch den ERC-1155 Standard anbietet, wird sich in dieser Arbeit darauf konzentriert [78].

8.2 Der ERC-1155 Token-Standard

Der ERC-1155 Standard, auch bekannt als "Multi Token Standard", ist eine innovative Erweiterung der Ethereum-Token-Standards. Er wurde entwickelt, um die Funktionalitäten von ERC-20 (für fungible Token) und ERC-721 (für nicht-fungible Token) in einem einzigen, effizienten Standard zu vereinen.

Besonderheiten des ERC-1155 Standards:

1. Multi-Token-Unterstützung: Ein einzelner Smart Contract kann verschiedene Token-Typen verwalten, sowohl funglible als auch nicht-fungible Token. Ein ERC-1155

Smart Contract verwaltet eine Kollektion an Token. Es können nach Demployment auf der Blockchain neue Token in diesem Smart Contract erstellt werden. ERC-1155 Token werden anhand der Adresse des Smart Contracts, in dem sie existieren und einer eindeutigen ID, die ihnen bei der Erstellung zugeschrieben wird, identifiziert.

2. Batch-Operationen: Der Standard ermöglicht die gleichzeitige Übertragung mehrerer Token-Typen in einer einzigen Transaktion.
3. Verbesserte Metadaten-Handhabung: Jeder Token-Typ kann mit eigenen Metadaten verknüpft werden, was eine flexiblere Darstellung von Assets ermöglicht. [79]

Fortlaufend werden die Begriffe *NFTs* und *Token* häufig verwendet. Im Kontext dieser Arbeit basieren beide auf dem ERC-1155 Standard. Traditionell bezeichneten NFTs (Non-Fungible Tokens) einzigartige, nicht-austauschbare digitale Assets, insbesondere Tokens mit medialem Inhalt. Der ERC-1155 Standard ermöglicht jedoch die Existenz mehrerer Exemplare eines Tokens, was die strikte Definition der “Nicht-Fungibilität” aufweicht. Zur Klarheit und Konsistenz werden in dieser Abhandlung die Begriffe *NFT* und *Token* synonym verwendet, um ERC-1155 Token zu beschreiben.

Medialer Inhalt auf IPFS

Das Hochladen von Medien auf dem InterPlanetary File System (IPFS) ist ein essentieller Schritt bei der Erstellung von NFTs. IPFS bietet eine dezentrale Möglichkeit, Dateien zu speichern und aufzurufen. Der Prozess beginnt damit, dass Dateien auf IPFS hochgeladen werden, wodurch ein einzigartiger Identifikator, der Content Identifier (CID), generiert wird. Dieser CID dient als Referenz in den Token-Metadaten [80].

Für ERC-1155 Token kann eine strukturierte Herangehensweise gewählt werden, bei der ein Basis-Ordner erstellt und auf IPFS hochgeladen wird. Innerhalb dieses Ordners können dann Unterordner mit spezifischen IDs angelegt werden, die die jeweiligen Medien und Metadaten für jeden Token in einer Kollektion enthalten. Diese Struktur ermöglicht es, einen Basis-IPFS-Pfad zu generieren und darauf aufbauend einzelne Verzeichnisse mit manuell eingetragenen Token-IDs zu erstellen. Dies ist besonders nützlich für ERC-1155 Token, da es die Verwaltung mehrerer Token-Typen innerhalb eines einzigen Smart Contracts erleichtert.

In den folgenden Kapiteln wird nicht detailliert auf die Erzeugung der Basis-URL und der spezifischen Token-URL eingegangen. In den dort vorgestellten Funktionsaufrufen werden diese beiden Parameter (`baseURL` und `tokenURL`) zwar dargestellt, jedoch wird der genaue Prozess ihrer Erstellung nicht weiter erläutert. Dies dient dazu, den Fokus auf die grundlegenden Konzepte und die Implementierung der Smart Contract-Funktionen zu legen, ohne sich in den technischen Details der IPFS-URL-Generierung zu verlieren. Die tatsächliche Erzeugung dieser URLs hängt in der Praxis von der gewählten IPFS-Implementierung und den spezifischen Projektanforderungen ab.

8.3 Looprings NFT Infrastruktur

Loopring hat für die Implementierung von NFT-Marktplätzen eine Infrastruktur entwickelt, die die Vorteile der L2-Technologie optimal nutzt. Im Zentrum dieser Infrastruktur stehen die L2 NFT-Funktionalitäten. Ein innovatives Konzept in diesem Zusammenhang sind die sogenannten Counterfactual NFTs.

8.3.1 Counterfactual NFTs

Das Konzept der Counterfactual NFTs ermöglicht es, NFTs auf L2 zu erstellen und zu handeln, ohne dass der entsprechende Smart Contract sofort auf L1 veröffentlicht werden muss. Dies spart Gaskosten und erhöht die Effizienz. Der tatsächliche NFT-Contract wird erst dann auf L1 veröffentlicht, wenn es notwendig wird, z.B. wenn jemand das NFT auf L1 transferieren möchte. Die Adresse des NFT-Contracts kann vor Deployment mit den Parametern der Adresse der NFT-Contract-Factory auf L1 und der des NFT-Besitzers deterministisch berechnet werden. Dies ermöglicht es, mit dem NFT zu interagieren, als ob der Smart Contract bereits existieren würde. Dieser Ansatz ist ein wesentlicher Teil von Loopring's Strategie, um die Kosten und Komplexität des NFT-Handels zu reduzieren und gleichzeitig die Sicherheit und Interoperabilität mit dem Ethereum-Mainnet beizubehalten [81].

8.3.2 Layer 2 Minting

Der Minting-Prozess auf Loopring beginnt somit vollständig auf L2. Wenn ein Benutzer ein NFT mintet, werden alle relevanten Informationen und Transaktionen auf L2 gespeichert und verarbeitet. Der tatsächliche Smart Contract auf L1 wird erst dann veröffentlicht, wenn es notwendig wird. Bis zu diesem Zeitpunkt existiert der Contract nur *counterfactual*, also als eine Art Versprechen oder Möglichkeit.
Die Architektur der L2 NFT Smart Contracts auf Loopring könnte verschiedene Formen annehmen. Der spezifische Code der Implementierung ist leider nicht öffentlich einsehbar. Loopring wird wahrscheinlich innovative und proprietäre Lösungen entwickelt haben, die speziell auf ihre L2-Infrastruktur zugeschnitten sind.

8.4 Architektur eines NFT-Marktplatzes auf Smart Contract Basis

Um das Implementierungskonzept eines NFT-Marktplatzes auf Taiko zu veranschaulichen, wurde für diese Arbeit eine Kombination aus einem Marktplatz-Smart-Contract (C), einem Token-Smart-Contract (B) und einem Factory-Smart-Contract (A) entwickelt,

die im Anhang zu finden ist. Die Token-Smart-Contracts sind für die eigentliche Verwaltung der NFTs bzw. Token verantwortlich, einschließlich der Speicherung von Besitzinformationen und der Durchführung von Transfers. Der Factory-Smart-Contract dient als zentraler Punkt für die Erstellung neuer Token-Smart-Contracts.
Die folgenden Konzepte wurden verwendet, um im Kapitel 11.2 eine mögliche Implementierung auf Taiko zu betrachten.

8.4.1 Der Factory-Smart-Contract

Wenn ein Nutzer einen neuen Token-Smart-Contract bzw. eine neue Token-Kollektion erstellen möchte, interagiert er mit der Factory. Dieser Smart Contract erstellt einen neuen ERC-1155 Token-Smart-Contract für eine neue Kollektion, in der neue Token erzeugt werden können.

Dieser Prozess läuft typischerweise so ab:

1. Ein Benutzer sendet einen Auftrag an den Factory-Smart-Contract, indem er dessen `createTokenContract` -Funktion aufruft. Dieser Auftrag kann nur für die Erstellung einer neuen Kollektion sein, oder auch direkt einen Mint-Auftrag für einen neuen Token beinhalten.
2. Der Factory-Smart-Contract erstellt und veröffentlicht einen neuen ERC-1155 Token-Smart-Contract.
3. Der Factory-Smart-Contract kann dabei die angeforderten Token direkt in dem neu erstellten ERC-1155 Token-Smart-Contract erzeugen.
4. Die öffentliche Adresse des neuen Token-Smart-Contracts wird gespeichert und dem Benutzer zurück gegeben.

Die Verwendung einer Factory in Verbindung mit individuellen ERC-1155 Smart Contracts bietet erhebliche Vorteile für die Benutzerfreundlichkeit. Dieser Ansatz ermöglicht es, dass jede Token-Kollektion oder auch einzelne NFTs in separaten Smart Contracts verwaltet werden können, ohne dass der Nutzer selbst einen ERC-1155 Smart Contract schreiben und veröffentlichen muss. Die Factory fungiert als zentraler Zugangspunkt und abstrahiert die komplexe Logik der Smart-Contract-Erstellung und dessen Initialisierung. Dadurch kann ein einfaches und benutzerfreundliches Interface angeboten werden, über das Nutzer ihre NFTs bzw. Token erstellen und verwalten können.

8.4.2 Der Token-Smart-Contract

Der Token-Smart-Contract ist verantwortlich für die:

- Erstellung neuer Token und NFTs (Minting) innerhalb einer Kollektion
- Speicherung von Token-Metadaten

- Verwaltung von Token-Besitz
- Durchführung von Token-Transfers

Im Folgenden Abschnitt werden Beispielfunktionen erläutert, die in diesen Smart Contract implementiert werden und welche Funktionalität sie erfüllen. Dabei werden nicht alle Funktionsmodifikatoren des Solidity Codes aufgezählt.

```
1 function mint(address receiver, uint256 amount, string memory
    tokenURI, bytes memory data) returns (uint256 tokenId)
```

Code-Snippet 8.1: Solidity: mint-Funktion des Token-Contracts

- Erstellt einen neuen NFT oder eine Anzahl von neuen Token in diesem Smart Contract. Durch diese Funktion kann der Besitzer des Smart Contracts die Kollektion an Token in diesem Smart Contract erweitern.
- Parameter:
  - `address receiver` : die Wallet, die das neue NFT oder die neuen Token erhält.
  - `uint256 amount` : die Anzahl der zu erstellenden Token (im ERC-1155 Standard können mehrere Token der gleichen ID existieren).
  - `string memory tokenURI` : enthält den Verweis zu den Metadaten des Token.
  - `bytes memory data` : zusätzliche Daten, die bei der Transaktion übergeben werden können. Dies kann für verschiedene Zwecke genutzt werden, wie z.B. weitere Informationen oder Hooks. Dieser Parameter wird oft leer gelassen (“0x”).

  - `returns uint256 tokenId` : identifiziert den spezifischen Token, der erzeugt wurde. Im ERC-1155 Standard kann ein einzelner Contract mehrere unterschiedliche Token verwalten. Jeder Token hat dabei eine eindeutige ID.

```
1     function setURI(uint256 tokenId, string memory uri)
```

Code-Snippet 8.2: Solidity: setURI-Funktion des Token-Contracts

- Der Besitzer des Smart Contracts kann über diese Funktion den Metadaten-Uniform-Resource-Identifier (URI) für einen spezifischen Token festlegen.
- Metadaten in NFTs enthalten essentielle Informationen wie Name, Beschreibung, Bild und Attribute des digitalen Assets, die typischerweise off-chain (auf IPFS) gespeichert und über einen URI im Smart Contract referenziert werden.
- Parameter:

  - `uint256 tokenId` : repräsentiert die ID des Token, dessen URI aktualisiert werden soll. Dies ermöglicht die Zuweising unterschiedlicher URIs zu verschiedenen Token im selben Smart Contract.
  - `string memory uri` : der URI-String, der die Metadaten für den spezifizierten Token enthält. Dies kann ein vollständiger URL, ein IPFS-Hash oder ein relativer Pfad sein. Dieser Verweis bestimmt, wo die Metadaten für den Token abgerufen werden können.

```
1    function setApprovalForAll(address operator, bool
       approved)
```

Code-Snippet 8.3: Solidity: setApprovalForAll-Funktion des Token-Contracts

- Erteilt oder entzieht einem Operator die Genehmigung, alle Token des Funktionsaufrufers, die in diesem ERC-1155 Smart Contract existieren, zu verwalten [82].
- Parameter:

  - `address operator` : die Adresse des Kontos (oder Smart Contracts), dem die Berechtigung erteilt oder entzogen werden soll, im Namen des Funktionsaufrufers zu agieren.
  - `bool approved` : dieser Parameter legt fest, ob die Genehmigung erteilt oder entzogen wird.

```
1    function safeBatchTransferFrom(address from, address to,
       uint256[] calldata ids, uint256[] calldata values,
       bytes calldata data)
```

Code-Snippet 8.4: Solidity: mint-Funktion des Token-Contracts

- Ermöglicht die Übertragung mehrerer unterschiedlicher Token in einer einzigen Transaktion [83].
- Parameter:

  - `address from` : die Adresse, von der die Token übertragen werden sollen. Dies muss der aktuelle Besitzer der Token oder eine genehmigte Adresse (`operator`) sein.
  - `address to` : die Empfängeradresse der Token.
  - `uint256[] calldata ids` : ein Array von Token-IDs, die übertragen werden sollen. Jede ID repräsentiert einen spezifischen Token.
  - `uint256[] calldata amounts` : ein Array mit der Menge der zu übertragenen Token. Dieses Array muss die gleiche Länge wie das `ids` Array haben. Jeder Eintrag entspricht der Menge des Token an der gleichen Position (Index) im `ids` Array.

  - `bytes calldata data` : zusätzliche Daten (optional).

```
function safeTransferFrom(address from, address to,
    uint256 id, uint256 amount, bytes calldata data)
```

Code-Snippet 8.5: Solidity: safeTransferFrom-Funktion des Token-Contracts

- überträgt eine bestimmte Menge eines spezifischen Token von einem Absender zu einem Empfänger [84].
- Parameter:
  - `address from` : die Adresse, von der die Token übertragen werden. Dies muss der aktuelle Besitzer oder ein Operator sein.
  - `address to` : die Empfängeradresse der Token.
  - `uint256 id` : die ID des Token, der übertragen werden soll.
  - `uint256 amount` : die Anzahl der zu übertragenen Token.
  - `bytes calldata data` : zusäzliche Daten (optional).

8.4.3 Der Marktplatz-Smart-Contract

Der Marktplatz-Smart-Contract verwaltet alle Aspekte des Marktplatzes, einschließlich Listings, Käufe und Verkäufe. Er interagiert mit dem Token-Smart-Contract und handhabt die gesamte Geschäftslogik des Marktplatzes. Die wichtigsten Funktionen umfassen:

- Erstellung und Verwaltung von Verkaufsangeboten (Listings)
- Stornieren von Angeboten
- Durchführung von Käufen
- Verwalten von Gebühren
- Speichern und Abrufen von Marktdaten

Beispielfunktionen in diesem Smart Contract sind:

```
function createListing(address tokenContract, uint256
    tokenId, uint256 price, uint256 quantity) returns (uint256
    listingId)
```

Code-Snippet 8.6: Solidity: createListing-Funktion des Token-Contracts

- Erstellt ein neues Verkaufsangebot für einen spezifischen Token mit festgelegtem Preis und Menge. Außerdem überprüft sie den Token-Besitz und Operator-Genehmigung des Marktplatzes über die ERC-1155 Operationsfunktionen.

- Parameter:
  - `address tokenContract` : die Adresse des Smart Contracts, in dem der zu verkaufende Token existiert.
  - `uint256 tokenId` : die ID des zu verkaufenden Token.
  - `uint256 price` : der Preis pro Token in der Währung der Plattform.
  - `uint256 quantity` : die Anzahl der zum Verkauf stehenden Einheiten.
  - `returns uint256 listingId` : gibt eine ID für das erstellte Listing zurück.

```
1    function cancelListing(uint256 listingsId)
```

Code-Snippet 8.7: Solidity: cancelListing-Funktion des Token-Contracts

- Ermöglicht es dem Verkäufer, ein bestehendes Verkaufsangebot zu stornieren.
- Parameter:
  - `uint256 listingsId` : die eindeutige ID des zu sornierenden Listings.

```
1    function buyNFT (uint256 listingsId, uint256 quantity)
       payable
```

Code-Snippet 8.8: Solidity: buyNFT-Funktion des Token-Contracts

- Führt den Kauf eines NFTs bzw. Token durch. Dabei handhabt sie die Bezahlung, aktualisiert die im Listing verfügbare Menge der Token, überträgt den Besitz der Token und emittiert ein Event, das die Transaktionsdaten aufzeichnet.
- Parameter:
  - `uint256 listingsId` : die ID des Listings, von dem gekauft wird.
  - `uint256 quantity` : die Anzahl der zu kaufenden Token.
  - `payable` : durch dieses Attribut kann Ether mit dem Aufrufen dieser Funktion gesendet werden.

```
1    function setMarketplaceFee(uint256 newFeePercentage)
```

Code-Snippet 8.9: Solidity: setMarketplaceFee-Funktion des Token-Contracts

- Erlaubt dem Besitzer des Marktplatzes, die Gebühren der Nutzung anzupassen.
- Parameter:

- `uint256 newFeePercentage` : der neue Prozentsatz für die Marktplatzgebühr.

```
function getListingsDetails(uint256 listingsId) returns
   (Listing memory)
```

Code-Snippet 8.10: Solidity: getListingsDetails-Funktion des Token-Contracts

- Ermöglicht es, detaillierte Informationen über ein spezifisches Listing abzurufen.
- Parameter:
  - `unit256 listingsId` : die ID des abzurufenden Listings.
  - `returns Listing memory` : gibt eine Struktur zurück, die alle wichtigen Informationen wie z.B. den Preis, die Anzahl etc. des zum Verkauf stehenden Token erfasst.

8.4.4 Datenstruktur im Marktplatz-Smart-Contract

Neben den grundlegenden Funktionen sind bestimmte Variablen und Datenstrukturen esenziell für eine effiziente Organisation und Verwaltung eines Marktplatzes. Folgende Implementationen ermöglichen die Speicherung und den schnellen Zugriff auf wichtige Informationen:

Die Verwaltung von Listings

```
struct Listing {
    uint256 listingId;
    address seller;
    address tokenContract;
    uint256 tokenId;
    uint256 price;
    uint256 quantity;
    bool isActive;
    uint256 timestamp;
}
mapping(uint256 => Listing) public listings;
uint256 private listingCounter;
```

Code-Snippet 8.11: Solidity: Struktur und Mapping für NFT Listings

Die `Listing` -Struktur definiert alle relevanten Details eines einzelnen Angebots im Marktplatz. Diese Listings werden in einem `mapping(uint256 => Listing)` gespeichert, das jeder eindeutigen Listing-ID die entsprechende Listing-Struktur zuordnet. Der `listingCounter` fungiert dabei als fortlaufender Zähler, der bei der Erstellung jedes

neuen Listings inkrementiert wird, um eine einzigartige ID zu generieren. Diese Kombination aus Struktur, Mapping und ID-Generator gewährleistet eine effiziente Speicherung, einen schnellen Zugriff und eine eindeutige Identifizierung jedes Listings im Marktplatz.

Angebote eines Verkäufer

```
1     mapping(address => EnumerableSet.UintSet) internal
          sellerListings;
```

Code-Snippet 8.12: Solidity: sellerListings-mapping; Angebote eines Verkäufers

Dieses Mapping verknüpft eine Verkäuferadresse mit einem Set aus IDs von dessen Listings, was eine schnelle Auflistung aller Angebote eines Verkäufers ermöglicht.

Ein *EnumerableSet.UintSet* in Solidity ist eine Datenstruktur aus der OpenZeppelin-Bibliothek, die ein Set von uint256-Werten verwaltet. Es ermöglicht effiziente Operationen wie Hinzufügen, Entfernen und Überprüfen der Existenz von Elementen in konstanter Zeit (O(1)), sowie die Aufzählung aller Elemente in linearer Zeit (O(n)), wobei keine Garantie für eine bestimmte Reihenfolge gegeben wird [85].

Token-Listing-Verwaltung

```
1     mapping(address => mapping(uint256 =>
          EnumerableSet.UintSet)) internal assignTokenToListings;
```

Code-Snippet 8.13: Solidity: assignTokenToListings-mapping; Bezug von Token zu Listings

Das `assignTokenToListings` Mapping nutzt eine verschachtelte Struktur, bei der ein Mapping in einem anderen Mapping eingebettet ist. Das äußere Mapping verwendet die Token-Contract-Adresse als Schlüssel und verweist auf ein weiteres, inneres Mapping. Dieses innere Mapping nutzt die Token-ID als Schlüssel und verknüpft diese mit einem Set von Listing-IDs. Durch diese zweistufige Verschachtelung kann der Marktplatz effizient Listings nach spezifischen Token-Contracts und Token-IDs organisieren und abrufen.

Marktplatz-Konfiguration

```
1     uint256 public marketplaceFeePercentage;
2     address public owner;
```

Code-Snippet 8.14: Solidity: Marktplatz Konfiguration

Diese Variablen speichern die Marktplatzgebühr und den Eigentümer des Marktplatzes, an den die Gebühr ausgezahlt wird.

Aktivitätsverfolgung

```
1     EnumerableSet.UintSet internal activeListings;
```

Code-Snippet 8.15: Solidity: activeListings-Set; Aktivitätsverfolgung

Dieses Set speichert die IDs aller aktiven Angebote. Die Verwendung eines Enumerable Sets bietet mehrere Vorteile gegenüber herkömmlichen Arrays oder Mappings. Es ermöglicht eine effiziente Verwaltung aktiver Listings mit konstanter Zeitkomplexität O(1) für die Verwaltung von aktiven Listings. Das bedeutet, dass diese Operationen unabhängig von der Anzahl der aktiven Listings immer die gleiche Zeit benötigen. Dadurch kann der Marktplatz auch bei einer großen Anzahl von Angeboten schnell und ressourcenschonend arbeiten. Die Geschwindigkeit dieser Operationen bleibt konstant, egal ob der Marktplatz 10 oder 10.000 aktive Listings hat. Zusätzlich bietet das Enumerable Set die Möglichkeit, effizient über alle aktiven Listings zu iterieren, was bei der Implementierung von Suchfunktionen oder der Anzeige von Angeboten nützlich ist.

Transaktionshistorie

```
1     event TokenSold(
2         uint256 indexed listingId,
3         address indexed tokenContract,
4         address indexed seller,
5         address buyer,
6         uint256 tokenId,
7         uint256 quantity,
8         uint256 price,
9         uint256 mfee,
10        uint256 timestamp
11        );
```

Code-Snippet 8.16: Solidity: TokenSold-event; Transaktionshistorie

Die Aufzeichnung der Transaktionshistorie des Marktplatz-Smart-Contract wird durch die Emission des `TokenSold`-Events realisiert. Dieses Event ergänzt die Standard-Events des ERC-1155 Contracts, um zusätzliche marktplatzspezifische Informationen bereitzustellen.
Der ERC-1155 Standard definiert bereits wichtige Events wie `TransferSingle` [86] und `TransferBatch` [87], die bei Token-Transfers emittiert werden. Diese Events enthalten grundlegende Informationen wie die beteiligten Adressen, Token-IDs und übertragene Mengen. Das `TokenSold` Event wird allerdings vom Marktplatz-Smart-Contract emittiert und fügt marktplatzrelevante Details hinzu:

- `listingId`: Identifiziert das spezifische Listing im Marktplatz.
- `price` und `mfee`: Erfassen den gesamten Verkaufspreis und die Marktplatzgebühr.

- `timestamp`: Ermöglicht eine präzise zeitliche Einordnung der Transaktion.

Diese zusätzlichen Informationen ermöglichen eine detailliertere Verfolgung und Analyse der Marktplatzaktivitäten. Die Verwendung von indexierten Parametern (`indexed`) für `listingId`, `tokenContract` und `seller` erleichtert die effiziente Filterung und Suche in den Event-Logs.

Zugelassene Token-Smart-Contracts

```
1     mapping(address => bool) public supportedTokenContracts;
```

Code-Snippet 8.17: Solidity: supportedContracts-mapping; zugelassene Token-Contracts

Das `supportedTokenContracts` Mapping verknüpft die Ethereum-Adressen von Token-Contracts mit einem booleschen Wert, der ihre Unterstützung anzeigt. Diese Struktur ermöglicht eine effiziente Überprüfung und Kontrolle darüber, welche Token auf dem Marktplatz gehandelt werden dürfen.

9 Vergleich der Skalierungsansätze

9.1 Loopring: Spezialisierte Skalierung durch vordefinierte Funktionalitäten

Loopring ähnelt in seiner Funktionsweise eher einer dezentralen Anwendung (dApp). Das Protokoll bietet spezifische API-Endpunkte und damit vordefinierte Funktionalitäten.

Layer 3 - Multi-Network-Unterstüzung
Looprings zukunftsorientierter Skalierungsansatz zielt darauf ab, Multi-Network Unterstützung zu implementieren. Dabei sollen die Smart Contracts des Loopring Protokolls auf verschiedenen bestehenden L2 Protokollen veröffentlicht werden, wodurch Loopring effektiv als Layer 3 Lösung fungiert. Dies ermöglicht eine breitere Nutzbarkeit und potenziell höhere Skalierbarkeit über mehrere Netzwerke hinweg [74].

Counterfactual NFTs
Ein innovativer Aspekt im Loopring-Protokoll ist das Konzept von Counterfactual NFTs. Es ermöglicht eine kostengünstigere Erzeugung von NFTs, indem es die tatsächliche Erstellung des NFTs auf der Blockchain verzögert, bis es notwendig wird.

Transactions per Second (TPS)
Looprings TPS variieren theoretisch je nach Konfiguration. Mit aktivierter On-Chain Data Availability (OCDA) kann Loopring bis zu 2.025 TPS erreichen. Ohne OCDA, im sogenannten Validium-Modus, wird in mehreren Artikeln eine theoretische Durchsatzrate von bis zu 16.400 TPS angegeben [88]. Hier ist wichtig zu beachten, dass diese Zahlen wahrscheinlich aus dem Design-Konzept [89] abgeleitet und nicht direkt von Loopring als garantierte Leistungswerte angegeben wurden. Die tatsächliche Leistung kann je nach Netzwerkbedingungen variieren. Derzeit werden von Loopring selbst 2.025 TPS angegeben [26].

9.2 Taiko: Dezentrale Skalierung durch Inception Layers

Taiko hingegen verfolgt einen anderen Ansatz. Das Protokoll wird zukünftig durch eine Infrastruktur aus dezentralen Nodes betrieben. Taiko setzt auf das Inception Layer-Prinzip, ein neuartiger Ansatz zur Skalierung.

Inception Layers

Das Inception Layer-Prinzip basiert auf der Ethereum-Äquivalenz von Taiko. Diese ermöglicht es, mehrere Layer ineinander zu verschachteln. Dies schafft eine Hierarchie von Blockchains, wobei jede Ebene von der Sicherheit der darunterliegenden Schicht profitiert. Dieser Ansatz verspricht eine exponentielle Steigerung der Skalierbarkeit, da jede zusätzliche Schicht die Gesamtkapazität des Systems erhöhen soll [90].

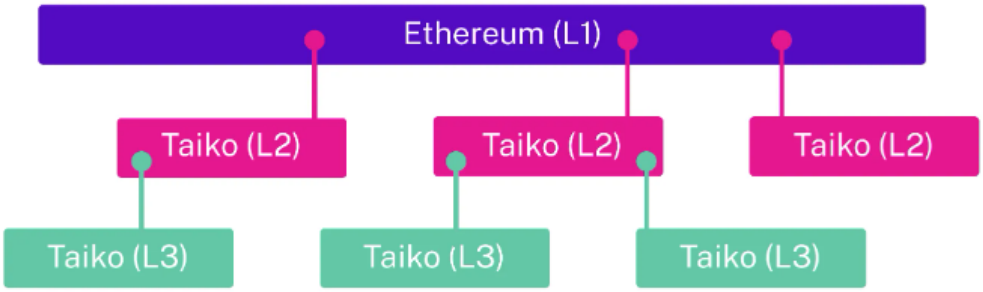


Abbildung 9.1: Inception Layer Prinzip
Quelle: [90]

NFT Minting

Für das Erstellen von NFTs auf Taiko wird, wie auf Ethereum L1, der Ansatz verfolgt, dass beim Minting eines NFTs eine Factory den entsprechenden Smart Contract erstellt oder in einem bestehenden Smart Contract neue Token erzeugt werden. Dieser Prozess ist weniger kosteneffizient als Looprings Counterfactual NFT Konzept.

Transactions per Day (TPD)

Die von Taiko veröffentlichten Statistiken zeigen die Metrik der Transaktionen pro Tag (TPD). Am 4. November 2024 erreichte die Plattform mit 5.001.802 Transaktionen einen Höchstwert, was etwa 58 TPS entspricht [91].

9.3 Gegenüberstellung: Skalierbarkeit

Kriterium	**Loopring**	**Taiko**
zukunftsorientierter Skalierungsansatz	Multi-Network-Unterstützung als Layer 3	Inception-Layer-Prinzip
TPS (OCDA)	Bis zu 2.025 TPS	58 TPS (potenziell höher)
TPS (Non-OCDA)	Bis zu 16.400 TPS (inoffizielle Angaben, basierend auf Designkonzepten)	genaue Zahlen nicht veröffentlicht
NFT Minting-Konzept	Counterfactual NFT-Prinzip	Smart Contract Deployment durch Factory

Tabelle 9.1: Vergleich der Skalierbarkeitsaspekte von Loopring und Taiko

Bewertung
Im Bereich der Skalierbarkeit hat Loopring derzeit eindeutig die Überhand. Dies liegt daran, dass zentralisierte Infrastrukturen in der Regel schnellere und effizientere Verarbeitungsgeschwindigkeiten und -kosten bieten können als dezentrale Dienste. Obwohl Taiko durch die Implementierung mehrerer verschachtelter Inception Layer möglicherweise ähnliche TPS wie Loopring erreichen könnte, bleibt das Loopring-Protokoll in der Lage, auf der letzten Schicht integriert zu werden, wodurch es erneut seine Überlegenheit zurückgewinnt.
Außerdem sind die Bestätigungszeiten auf Loopring schneller, da das Abwickeln der Transaktionen nicht von den Validatoren des Ethereum-Netzwerks abhängt, sondern ausschließlich vom zentralen Relayer.

10 Ansätze einer NFT-Marktplatz-Implementierung

Die Analyse der Kernkomponenten und Prozesse in Kapitel 8.4 soll das Fundament für das Verständis der Implementierungen auf Loopring und Taiko bilden. Als nächster Schritt in diesem Vergleichsprozess wird die Untersuchung der spezifischen Entwicklerumgebungen und -tools dieser beiden L2-Lösungen folgen.
In der Gegenüberstellung der generellen Infrastruktur der beiden Rollups zeigten sich bereits eindeutige Unterschiede in der Kommunikation zu beiden Netzwerken. Zusammenfassend lassen sich die beiden Ansätze der Skalierungslösung folgendermaßen beschreiben:

Loopring
Loopring bietet eine spezialisierte ZK-Rollup-Infrastruktur, die unter anderem für bestimmte NFT-Operationen optimiert ist. Diese Infrastruktur umfasst vorgefertigte API-Endpunkte, die es Entwicklern ermöglichen, NFT-Minting, Listing, Kaufen und Verkaufen effizient umzusetzen. Ein integriertes Orderbook-System unterstützt die Verwaltung und den Handel von NFTs, während die optimierte ZK-Rollup-Technologie schnelle Verifizierungen und erhöhten Datenschutz gewährleistet. Loopring legt großen Wert auf Skalierbarkeit und Sicherheit, was sich in den vorhandenen Maßnahmen widerspiegelt. Allerdings bietet Loopring aufgrund seiner limitierten Infrastruktur wenig Anpassungsmöglichkeiten für einen NFT-Marktplatz. Dies schränkt die Flexibilität und Innovationsmöglichkeiten für Entwickler ein, die spezifische oder fortgeschrittene Funktionen implementieren möchten. Derzeit wird nur ein aktiver Relayer betrieben, und zwar vom Loopring-Team selbst, was eine gewisse Zentralisierung und somit Abhängigkeit von einem einzelnen Dienstleister innerhalb des Systems bedeutet.

Taiko
Im Gegensatz dazu bietet Taiko eine offene EVM-kompatible Umgebung mit hybrider Rollup-Technologie. Diese Umgebung ermöglicht die Nutzung einer Typ 1 zkVM mit vollständiger Ethereum-Äquivalenz. Taiko legt besonderen Wert auf Sicherheit und Dezentralisierung, was sich in der entworfenen Infrastruktur mit Proposern und Provern widerspiegelt. Jeder Netzwerkteilnehmer kann in Zukunft ohne Genehmigung eine dieser Validatoren betreiben, wodurch die Dezentralität weiter gestärkt wird. Nutzer haben die Möglichkeit, eigene Smart Contracts zu verwalten und über RPC-Endpunkte auf Low-Level-Funktionen der Blockchain zuzugreifen. Dies bedeutet jedoch auch, dass die komplette Infrastruktur eines NFT-Marktplatzes selbst erstellt und die Smart Contracts auditiert werden müssen, was einen erheblichen Aufwand darstellen kann.

10.1 Ansatz auf Loopring

Nutzung der API und des Loopring SDKs
Die Implementation eines NFT-Marktplatzes auf Loopring basiert maßgeblich auf der Nutzung des Loopring SDK und der vordefinierten API-Funktionen. Das Loopring SDK bietet eine umfassende Sammlung von Werkzeugen zur Integration von Wallet-Verbindungen und zum Transaktionsmanagement, welches vom Loopring-Team bereitgestellt wird.

Herausforderungen mit dem Loopring SDK
In der Praxis zeigt sich jedoch, dass die Nutzung des offiziellen SDKs mit einigen Herausforderungen verbunden ist. Insbesondere bei der Implementierung in TypeScript treten häufiger Fehler auf, die auf Inkompatibilitäten oder unzureichende Dokumentation zurückzuführen sind. Die bereitgestellte Dokumentation des SDKs besteht größtenteils aus Codebeispielen, die zwar einzelne Funktionen demonstrieren, jedoch keine umfassenden Erklärungen oder einen ganzheitlichen Überblick bieten [92]. Solche Probleme können dazu führen, dass spezifische Funktionalitäten nicht wie erwartet funktionieren und Fehlerbehebungen sehr mühsam sind. In diesen Fällen kann es notwendig werden, Funktionen manuell zu implementieren oder den vorhandenen Code aus dem SDK als Grundlage zu verwenden, um ihn individuell anzupassen. In dieser Arbeit wird von einer Analyse der SDK-Funkionalitäten abgesehen. Es wird nur auf die Endpunkte der API eingegangen.

Looprings Signaturprozess
Ein kritischer Aspekt bei einer Implementierung ist der spezielle Signaturprozess, der von Loopring vorgegeben wird. Dieser Prozess, der spezifische Parameter erfordert, muss korrekt implementiert werden, damit die API-Anfragen akzeptiert werden. Dies stellt eine zusätzliche Komplexitätsebene dar, die bei der Entwicklung berücksichtigt werden muss. Es ist jedoch wichtig zu betonen, dass dieser Aspekt nur eine Rolle spielt, wenn selbst eine Wallet-Applikation erstellt werden soll. Falls dies nicht der Fall ist, kann für den Signaturprozess theoretisch das Loopring SDK, die Loopring Wallet oder ein Plugin für MetaMask [93] verwendet werden.

10.2 Ansatz auf Taiko

Entwicklung und Deployment von Smart Contracts
Auf Taiko liegt der Schwerpunkt auf dem Design und der Entwicklung eigener Smart Contracts, um die spezifischen Anforderungen an NFT-Funktionalitäten zu erfüllen. Anders als auf Loopring, wo ein integriertes Orderbook-System verwendet wird, muss auf Taiko ein eigener Marktplatz-Smart-Contract entwickelt werden, der die Angebote auflistet. Zusätzlich muss ein Factory-Smart-Contract und der ERC-1155 Token-Smart-Contract entworfen werden, um NFTs erzeugen zu können und alle Funktionalitäten

abzudecken.

Verwendung von ethers.js v6

Die Bibliothek ethers.js wird verwendet, um mit den entworfenen Smart Contracts zu interagieren. Sie ermöglicht es, vordefinierte Instanzen zu nutzen, die eine Reihe an Funktionalitäten bieten. Eine Wallet-Instanz wird dazu verwendet, Transaktionen zu signieren und an das Netzwerk zu senden. Über Contract-Instanzen können vordefinierte Funktionen der Smart Contracts aufgerufen werden. Ethers.js v6 vereinfacht den Prozess des Lesens von Blockchain-Daten und des Schreibens neuer Informationen durch Transaktionen. Es bietet eine Schnittstelle für die Kommunikation zwischen dem Frontend des NFT-Marktplatzes und Taiko, wodurch komplexe Interaktionen wie das Minten, Kaufen und Verkaufen von NFTs effizient umgesetzt werden können [94].

RPC-Endpunkte

Die Implementierung eines NFT-Marktplatzes auf Taiko erfordert die Integration der Taiko RPC-Endpunkte, die als Schnittstelle zwischen dem Frontend und der Blockchain dienen. Taiko bietet öffentliche RPC-URLs an, die für die Interaktion mit dem Netzwerk genutzt werden können. Über diese Endpunkte werden Transaktionen gesendet, Blockchain-Daten abgefragt und mit Smart Contracts interagiert.
Die öffentlichen RPC-URLs von Taiko unterliegen einer Ratenbegrenzung. Für Produktionsanwendungen oder bei hohen Nutzerzahlen könnte es notwendig sein, eine eigene Proposer-Node zu betreiben oder zukünftig alternative RPC-Provider (wie z.B. Infura) in Betracht zu ziehen, um eine zuverlässige Leistung sicherzustellen. Derzeit bietet allerdings nur Taiko einen öffentlichen RPC-Endpunkt an [73].

11 Betrachtung des Entwicklungsaufwands

Dieses Kapitel betrachtet potenzielle Umsetzungen eines NFT-Marktplatzes auf beiden Skalierungslösungen. Der Fokus liegt dabei auf der Interaktion mit den Backend- bzw. Blockchain-Prozessen und den spezifischen Anforderungen jedes Rollups, um die Kernfunktionen Minten, Kaufen und Verkaufen umzusetzen. Im Rahmen der folgenden Analyse werden Konzepte für Frontend-Funktionen und -Implementierungen in der Programmiersprache JavaScript dargestellt. Brückenfunktionalitäten bzw. Registrierungs- und Einzahlungsprozesse auf Layer 2 sowie der *detaillierte* Signaturprozess für Loopring werden in dieser Analyse nicht berücksichtigt.
Der dargestellte Code dient zum Verständis der unterschiedlichen Interaktionsweise und Infrastruktur der Implementierungen auf beiden Skalierungslösungen. Der selbst erstellte Solidity-Code wurde in REMIX getestet, allerdings nicht umfangreich von mehreren Parteien analysiert oder auditiert.

11.1 Implementierungskonzept auf Loopring

Diese Untersuchung konzentriert sich primär auf die API-Anfragen und Parameter, die für die Umsetzung der Kernfunktionen auf Loopring erforderlich sind. Der JavaScript Code soll lediglich Konzepte widerspiegeln, wie die betrachteten Funktionalitäten umgesetzt werden könnten. Dabei wird von einer Betrachtung des Loopring SDK abgesehen. Eine funktionsfähige Implementierung könnte aus diversen Gründen von dem dargestellten Code abweichen.

11.1.1 EdDSA-Schlüssel und -Signatur

Die Interaktion mit dem Loopring-Protokoll basiert auf einem EdDSA-Schlüsselpaar, welches für jede Ethereum-Adresse, die sich auf Loopring registriert, einmalig erstellt wird. Dieses Schlüsselpaar, bestehend aus einem öffentlichen und einem privaten Schlüssel, dient der Signierung von Transaktionen und der Gewährleistung der Authentizität von API-Anfragen. Das Schlüsselpaar wird über den Edwards-curve Digital Signature Algorithm (EdDSA) und einer spezifischen elliptische Kurve berechnet.
Die Ethereum-Adresse des Nutzers wird verwendet, um das Loopring-Konto zu identifizieren und das EdDSA-Schlüsselpaar diesem Konto auf L2 zuzuordnen. Somit ist ein Konto auf Loopring mit der selben öffentlichen Adresse wie auf Ethereum L1 ansprechbar. Dies ermöglicht es Loopring, die Sicherheit und Effizienz von EdDSA für L2-Transaktionen zu nutzen, während die Verbindung zur Ethereum-Main-Chain beibehalten wird [37, NOTE]. Der private EdDSA-Schlüssel wird für die klientenseitige digitale Signatur von API-Anfragen verwendet, während der öffentliche Schlüssel in der Merkle-

Baumstruktur von Loopring gespeichert und zur Verifizierung der Signaturen eingesetzt wird [95].

Die Erstellung einer EdDSA-Signatur für Loopring auf der Klientenseite ist ein komplexer und rechenintensiver Prozess, der spezifisches kryptographisches Fachwissen erfordert. Dieser Vorgang basiert auf der Verwendung der JubJub [96] bzw. BabyJub [97] Kurve, die für Zero-Knowledge-Proof-Systeme optimiert wurde [98]. Der Prozess beinhaltet mehrere anspruchsvolle Schritte, darunter das Hashing der zu signierenden Nachricht mit dem Poseidon-Algorithmus und komplexe mathematische Operationen auf der elliptischen Kurve. Insbesondere die Kurvenmultiplikationen sind rechenintensiv und erfordern eine sorgfältige Implementierung [99]. Die Kombination dieser Elemente macht die EdDSA-Signaturerstellung zu einem aufwendigen Verfahren, das präzise umgesetzt werden muss, um die Sicherheit und Effizienz des Loopring-Protokolls zu nutzen.

11.1.2 API-Schlüssel

Zusätzlich benötigt jedes Konto einen API-Schlüssel für die Authentifizierung bei API-Anfragen. Er wird vom Loopring-Relayer generiert und ist an das Benutzerkonto gebunden. Der API-Schlüssel muss bei jeder API-Anfrage als HTTP-Header-Wert übermittelt werden, um die Anfrage zu authentifizieren. Sie können bei Bedarf aktualisiert werden [100].

11.1.3 Minting von NFTs

Bei der Erzeugung von NFTs basierend auf dem ERC-1155 Standard gibt es 2 Szenarien zu beachten. Die Erstellung eines neuen Smart Contracts (Kollektion) oder die Erstellung von NFTs in einer bereits existierenden Kollektion.

Erstellung einer Kollektion
Die Erstellung einer Kollektion erfolgt über den API-Endpunkt POST /api/v3/nft/collection [101]. Ein Code-Beispiel für die Interaktion mit dem Relayer von Loopring könnte folgendermaßen strukturiert sein:

```
1 const axios = require('axios');
2
3 async function createCollection() {
4   const url =
      'https://api3.loopring.io/api/v3/nft/collection';
5   const headers = {
6     X-API-KEY: 'user_api_key_here',
7     X-API-SIG: 'user_signature_here',
8     Content-Type: 'application/json'
9   };
```

```
10
11   //weiterer Code...
```

Code-Snippet 11.1: JavaScript: createCollection-Funktion auf Loopring I

In diesen Beispielen wird die Javascript-Bibliothek `axios` für HTTP-Anfragen verwendet. Sie vereinfacht die Durchführung von asynchronen Netzwerkanfragen wie GET, POST, PUT und DELETE, indem sie eine Promise-basierte API bereitstellt und automatisch JSON-Daten verarbeitet. In einer tatsächlichen Implementierung würde höchst wahrscheinlich das Loopring SDK verwendet werden.

Als erstes wird die `headers` Komponente für die API-Anfrage betrachtet. Die Verwendung von X- als Präfix für benutzerdefinierte HTTP-Header ist eine gängige Praxis, die dazu dient, benutzerdefinierte Header von standardisierten HTTP-Headern zu unterscheiden. Loopring gibt diese Formatierung für ihre API vor.

`X-API-KEY`	ist der API-Schlüssel zur Authentifizierung eines Benutzers.
`X-API-SIG`	ist die EdDSA-Signatur, die mit dem privaten EdDSA-Schlüssel des Benutzers erstellt wird. **Anmerkung**: Loopring weist keine einheitliche Formatierung der API vor. Bei manchen Anfragen muss die Signatur im Header angegeben werden, bei anderen Anfragen ist sie mit im Request-Data-Objekt enthalten (Stand Dezember 2024). Zum Verständis des Codes wird im Folgenden immer davon ausgegangen, dass die Signatur im HTTP-Header enthalten ist. Dies könnte in einer tatsächlichen Implementierung abweichen.
`Content-Type`	gibt das Format der gesendeten Daten an, typischerweise application/json für JSON-formatierte Anfragen.

Im nächsten Code-Block wird die Funktion um die zu übermittelnden Daten erweitert:

```
1  const axios = require('axios');
2
3  async function createCollection() {
4    const url =
       'https://api3.loopring.io/api/v3/nft/collection';
5    const headers = {
6      X-API-KEY: 'user_api_key_here',
7      X-API-SIG: 'user_signature_here',
8      Content-Type: 'application/json'
9    };
10
11   const data = {
```

```
    name: "test",
    owner: "0x1cACC96e5F01e2849E6036F25531A9A064D2FB5f",
    nftFactory: "0x25315F9878BA07221db684b7ad3676502E914894",
    tileUri: "ipfs://QmfS4WckWPzn2eNU53zuTM9qYyjUpwWYLK..."
  };

  //weiterer Code...
```

Code-Snippet 11.2: JavaScript: createCollection-Funktion auf Loopring II

Dieses data-Objekt wurde aus dem Beispiel-Code aus Looprings Dokumentation entnommen [102]. Darin sind 4 notwendige Parameter enthalten.

Erklärung der Parameter im `data`-Objekt [101]:

`name`	beschreibt den Namen der Kollektion. Jeder Nutzer kann einen Namen nur einmalig verwenden.
`owner`	ist die Ethereum Adresse des Erstellers.
`nftFactory`	ist die Adresse der NFT-Factory von Loopring auf Layer 1. Sie erzeugt die Smart Contracts, sobald ein NFT auf Layer 1 transferiert werden soll.
`tileUri`	bezieht sich auf eine URI, die auf ein Bild verweist, das als Kachel oder Hintergrundbild für die NFT-Kollektion verwendet wird. Dieses Bild dient als visuelles Erkennungsmerkmal oder Repräsentation der gesamten Kollektion.

Conterfactual NFT Collections

Mit diesem Ansatz nutzt Loopring das Konzept der Counterfactual NFTs, um die zukünftige Adresse des Smart Contracts deterministisch zu berechnen. Mithilfe der `owner`-Adresse und der `nftFactory`-Adresse wird die Contract-Adresse vorausgesagt, bevor der Smart Contract tatsächlich auf Layer 1 erstellt wird [81].

Eine vollständige Funktion könnte mithilfe von `axios` über einen `try-catch` Block umgesetzt werden und folgende Struktur aufweisen:

```
const axios = require('axios');

async function createCollection() {
  const url =
     'https://api3.loopring.io/api/v3/nft/collection';
  const headers = {
    X-API-KEY: 'user_api_key_here',
    X-API-SIG: 'user_signature_here',
    Content-Type: 'application/json'
```

```
  };

  const data = {
    name: "test",
    owner: "0x1cACC96e5F01e2849E6036F25531A9A064D2FB5f",
    nftFactory: "0x25315F9878BA07221db684b7ad3676502E914894",
    tileUri: "ipfs://QmfS4WckWPzn2eNU53zuTM9qYyjUpwWYLKdA..."
  };
    try {
        const response = await axios.post(url, data, {
           headers });
        console.log('Kollektion erstellt. Die Adresse lautet:
           ', response.contractAddress);
        return response.contractAddress;
      } catch (error) {
        console.error('Error minting NFT:',
        error.response ? error.response.message :
           error.message);
      };
    };
}
```

Code-Snippet 11.3: JavaScript: createCollection-Funktion auf Loopring III

Es ist wichtig zu betonen, dass die variablen Parameter der `data`-Objekte eigentlich in die Funktionen übergeben werden müssten. In diesen Beispielen wird das `data`-Objekt in der Funktion definiert und mit Dummy-Daten gefüllt. In einer richtigen Implementierung würde ein Nutzer die variablen Parameter über ein Frontend Interface eintragen, sodass sie bei dem Funktionsaufruf übergeben und dynamisch im `data`-Objekt eingetragen werden.
Fortlaufend weisen alle Beispielfunktionen für die Implementierung auf Loopring generell eine ähnliche Struktur auf, indem ein API-Aufruf über einen `try-catch` Block umgesetzt wird und die Antwort des Relayers (`response`) verarbeitet wird. Die `response` Objekte unterscheiden sich in ihren Strukturen, sodass daraus je nach Art der Anfrage unterschiedliche Parameter extrahiert werden. Der `try-catch` Block der Funktionen wird fortlaufend nicht weiter beleuchtet.

Minting in einer bestehenden Kollektion

Die Implementierung des Minting-Prozesses von NFTs erfolgt über den API-Endpunkt POST /api/v3/nft/mint [78]. Dieser Endpunkt ermöglicht die Erstellung neuer NFTs direkt auf Layer 2 in einer vorhandenen Kollektion. Die folgende Funktion weist Unterschiede in der `url`-Konstante und dem `data`-Objekt zu der vorherigen Beispielfunktion auf. Für das `data`-Objekt wurde sich wieder an dem Sample-Code der Dokumentation orientiert [103]:

```
async function mintNFT() {
  const url = 'https://api3.loopring.io/api/v3/nft/mint';
  const headers = {...};

```

```
  const data = {
    exchange: "0x2e76EBd1c7c0C8e7c2B875b6d505a260C525d25e",
    minterId: 10010,
    minterAddress:
       "0x6b1029c9ae8aa5eea9e045e8ba3c93d380d5bdda",
    toAccountId: 10010,
    toAddress: "0x6b1029c9ae8aa5eea9e045e8ba3c93d380d5bdda",
    nftType: 0, // 0 for ERC1155
    tokenAddress:
       "0xc30e4cb0853470e6a412ca7f21363cd0d26428e8",
    nftId: "0x407294a8f36d3bdc4d04a6fce57afe0f8d49da56e0...",
    amount: "5",
    validUntil: 1655873983
    royaltyPercentage: 5,
    forceToMint: false,
    storageId: 1059,
    maxFee: {
        tokenId: 0, // 0 for ETH
        amount: '100000000000000' // 0.0001 ETH in Wei
        },
    }

//weiterer Code...
```

Code-Snippet 11.4: JavaScript: mint-Funktion auf Loopring

Erklärung der Parameter im `data`-Objekt [78]:

`exchange`	repräsentiert die Ethereum-Adresse des Loopring Exchange Smart Contracts auf Layer 1.
`minterId`	beschreibt die Loopring-interne ID des Accounts, der die Token erzeugt.
`minterAddress`	bezeichnet die Ethereum-Adresse, die den Token erzeugt.
`toAccountId`	ist die Loopring-interne ID des Empfängeraccounts für die erzeugten Token.
`toAddress`	beschreibt die Ethereum-Adresse des Empfängers der neuen Token.
`nftType`	ist ein numerischer Wert, der den Standard der Token festlegt. Der Wert 0 steht für den ERC-1155 Standard, den Loopring für seine Token verwendet.
`tokenAddress`	bezieht sich auf eine Kollektion (einen Counterfactual ERC-1155 Smart Contract) auf Loopring, in der der Token erzeugt werden soll.

`nftId`	identifiziert auf Loopring den Token und enthält gleichzeitig den Verweis zu den Metadaten auf IPFS, da sie direkt aus der IPFS Content Identifier (CID) generiert wird, wodurch eine eindeutige Verknüpfung zwischen dem Token auf der Blockchain und seinen off-chain Metadaten hergestellt wird [104].
`amount`	beschreibt die Anzahl des zu erstellenden Token.
`validUntil`	erfasst einen Zeitstempel, der angibt, bis zu welchem Zeitpunkt die Transaktion bzw. ein Auftrag ausgeführt werden soll. Nach überschreiten des Zeitstempel wird der Auftrag bzw. die Transaktion als ungültig betrachtet und nicht mehr ausgeführt. Dies verhindert, dass alte oder verzögerte Transaktionen zu einem späteren Zeitpunkt unerwartet ausgeführt werden.
`royaltyPercentage`	gibt an, wie hoch die Beteiligung des Erstellers des Token an einem Wiederverkauf ist. Im oberen Beispiel sind es 5 Prozent.
`storageId`	ist eine einzigartige 32-Bit-Zahl, die für jedes Konto und jeden Token spezifisch ist. Sie wird für verschiedene Operationen verwendet, einschließlich Transaktionen und Orders im Loopring-System. Der storageID wird im Merkle-Baum gespeichert und nach jeder relevanten Operation inkrementiert. Dies dient nicht nur zur Identifikation, sondern auch als Schutz gegen Replay-Attacken für bestimmte Transaktionstypen wie Spot-Trades und Transfers. Um die nächstmögliche ID zu erhalten, stellt Loopring einen speziellen API-Endpunkt zur Verfügung. Der storageID wird vom Loopring-Relayer vorgegeben und ist Teil der On-Chain-Daten [105, 106].
`maxFee`	besteht aus den beiden Gebührenparametern und beschreibt die maximale Gebühr, die ein Nutzer bereit ist zu zahlen, um die Token zu erzeugen.
`-tokenId`	ist ein numerischer Wert, der die Währung beschreibt, in der gezahlt wird (entweder Ether oder Loopring-Coin).
`-amount`	erfasst den maximalen Gebührenbetrag in der kleinsten Einheit der Währung (Wei).

Der Minting-Vorgang findet vollständig in der L2-Umgebung von Loopring statt. Nach dem Senden der Anfrage verarbeitet das Loopring-Protokoll die Transaktion off-chain auf dem Relayer. Dies ermöglicht eine schnelle und kostengünstige Erstellung der NFT. Die Transaktion wird später in einem Batch mit anderen Transaktionen auf die Ethereum-Main-Chain übertragen.

Nach erfolgreichem Minten ist das NFT in der Loopring L2-Wallet des Erstellers verfügbar und kann auf dem Loopring-Marktplatz gehandelt oder auf Ethereum L1 übertragen werden.

11.1.4 Verkaufen von NFTs (Erstellen von Listings)

Für das Erstellen von NFT-Verkaufsangeboten (Listing) wird der API-Endpunkt /api/v3/nft/validateOrder verwendet [107]. Dieser Endpunkt ermöglicht es, NFTs zum Verkauf anzubieten. Eine Implementierung könnte folgende Struktur aufweisen:

```
async function createNFTListing() {
  const url =
     'https://api3.loopring.io/api/v3/nft/validateOrder';
  const headers = {...};

  const data = {
    exchange: '0x0BABA1Ad5bE3a5C0a66E7ac838a129Bf948f1eA4',
    accountId: 12345,
    storageId: 12345,
    sellToken: {
      tokenId: 32769,
      nftData:
         '0xf7c932351186c3a9053f313eefa16209c018f7f1dba..',
      amount: '5'
    },
    buyToken: {
      tokenId: 0,  // 0 steht fuer ETH
      amount: '500000000000000000' //0.5 ETH (5 * 0.1 ETH pro
         NFT)
    },
    validUntil: 1656169134,
    maxFeeBips: 100,
    clientOrderId: 'MyNFTListing',
    affiliate: 0,
    fillAmountBOrS: false  // Erlaubt Teilerfuellungen
  };

//weiterer Code...
```

Code-Snippet 11.5: JavaScript: Erstellen von Listings auf Loopring

Auch hier wird ein `data`-Objekt erzeugt, dass die notwendigen Parameter enthält, damit der Relayer die Anfrage bearbeiten kann. In der Anfrage für das Erstellen eines Listings

sind folgende neue Parameter wichtig:

`sellToken`	beinhaltet 3 Parameter:
`-tokenId`	ist die Loopring-spezifische ID des Tokens/NFT.
`-nftData`	ist hier ein Hash aus der `tokenAddress` und der `nftId`.
`-amount`	ist die Anzahl der zu verkaufenden Token.

`buyToken`	beinhaltet 2 Parameter:
`-tokenId`	ist ein numerischer Wert, der angibt in welcher Währung gezahlt werden soll (0 steht für ETH).
`-amount`	gibt an, wieviel der Verkäufer für alle Token insgesamt erhalten möchte. Falls 5 Token zum Verkauf angeboten werden, wird die Gesamtsumme angegeben.

`maxFeeBips`	ist die maximal Gebühr, die ein Nutzer bereit ist, für die Erstellung des Listings zu bezahlen. Sie wird in Basispunkten (BIPs, 1 BIP = 0.01%) berechnet.

`clientOrderId`	dient zur eindeutigen Erkennung des Auftrags und wird vom Nutzer festgelegt. Die Angabe ist optional und kann leer gelassen werden.

`affiliate`	gibt die Loopring-Account-ID eines Affiliates an. Dies ermöglicht eine Gebührenbeteiligung eines Dritten (z.B. ein Marktplatzbetreiber). Die Gebühr wird auf Protokollebene festgelegt, sodass ein Marktplatzbetreiber mit dem Loopring Team Kontakt aufnehmen muss, um diese festzulegen. Falls kein Affiliate existiert, wird der Wert 0 eingetragen.

`fillAmountBOrS`	bestimmt, ob ein Auftrag teilweise oder nur vollständig erfüllt werden kann.

Sortieren von Listings über einen Indexer

Auf Loopring gibt es keine direkte Funktionalität, um alle existierenden Listings auf einmal abzurufen, da dies aus Skalierbarkeits- und Effizienzgründen wahrscheinlich problematisch wäre. Stattdessen könnte ein Indexer eine zentrale Rolle bei der Implementierung eines Marktplatzes auf Loopring spielen. Ein Indexer wäre ein Bestandteil, der kontinuierlich die Erstellung von Listings überwacht und diese Informationen in einer eigenen Datenbank speichert und indexiert. Dieser Prozess ermöglicht es dem Marktplatz, schnell und effizient auf die Daten der gelisteten NFTs zuzugreifen, ohne jedes Mal direkt mit der Loopring-Infrastruktur kommunizieren zu müssen. Laut der Dokumentation von Loopring gibt es leider keine Funktionalität der WebSocket-API, die Erstellung

oder Updates von Listings zu überwachen. Dies müsste also in die eigenen Funktionen `createNFTListing` und `buyNFT` eingebaut werden. Für das Frontend bedeutet dies, dass es zur Anzeige von Listings mit dem eigenen Indexer kommuniziert. Der Indexer stellt sicher, dass die Marktplatz-Implementierung über aktuelle Informationen zu den NFT-Listings verfügt. Er ermöglicht es dem Frontend, Listings zu filtern und zu sortieren, was die Benutzererfahrung erheblich verbessert.

11.1.5 Kaufen von NFTs

Um einen NFT auf Loopring zu kaufen, wird der API-Endpunkt POST api/v3/nft/trade verwendet [108]. Dieser Endpunkt bietet die Grundsturktur für verschiedene Handelsaufträge. Die folgende JavaScript Funktion implementiert ein `data`-Objekt mit einer neuen Struktur:

```
async function buyNFT() {
  const endpoint =
     'https://api3.loopring.io/api/v3/nft/trade';
  const headers = {...};

  const data = {
    maker: {
        exchange:
           "0x2e76EBd1c7c0C8e7c2B875b6d505a260C525d25e",
        accountId: 12454,
        storageId: 54,
        sellToken: {
            tokenId: 32768,
            nftData: "0x1a2001aac7a1fd00cef07889cdb...",
            amount: "1"
        },
        buyToken: {
            tokenId: 0,
            amount: "1000000000000000000"
        },
        allOrNone: false,
        fillAmountBOrS: false,
        validUntil: 1656227891,
        maxFeeBips: 1000,
    },
    makerFeeBips: 1000,
    taker: {
        exchange:
           "0x2e76EBd1c7c0C8e7c2B875b6d505a260C525d25e",
        accountId: 10488,
        storageId: 64,
        sellToken: {
            tokenId: 0,
            amount: "1000000000000000000"
        },
```

```
33          buyToken: {
34              tokenId: 32768,
35              nftData: "0x1a2001aac7a1fd00cef07889cdb...",
36              amount: "1"
37          },
38          allOrNone: false,
39          fillAmountBOrS: true,
40          validUntil: 1656227891,
41          maxFeeBips: 100,
42      },
43      takerFeeBips: 100
44  }
45    //weiterer Code...
```

Code-Snippet 11.6: JavaScript: Kaufen von NFTs auf Loopring

Das `data`-Objekt besteht bei diesem API-Endpunkt aus einem `maker`- und einem `taker`-Objekt.
Das `maker`-Objekt repräsentiert die Verkaufsseite des NFT-Handels und erfasst die Parameter, die bereits im Kapitel 11.1.4 erklärt wurden. Dazu kommt folgender Parameter:

`makerFeeBips` gibt die Gebühr an, die der Verkäufer für die Ausführung seines Handelsauftrags zahlen muss. Sie wird auch in Basispunkten (BIP) angegeben.

Das `taker`-Objekt repräsentiert die Käuferseite des NFT-Handels und erfasst die gleichen Parameter wie das `maker`-Objekt. Der Unterschied ist, dass `sellToken` und `buyToken` umgekehrt aufgebaut sind, da sie die Perspektive der anderen Seite des Handels beschreiben. Dazu kommt der Parameter:

`takerFeeBips` gibt die Gebühr an, die der Käufer für die Ausführung seines Handelsauftrags zahlen muss.

Während der Endpunkt hauptsächlich für den Kauf von bereits gelisteten NFTs konzipiert ist, bietet er theoretisch auch die Flexibilität, je nach Implementierung des Marktplatzes, für Gebote genutzt zu werden. Diese Anpassungsfähigkeit erlaubt es Entwicklern, den Endpunkt für verschiedene Handelsmodelle zu nutzen, sei es für Festpreis-Listings oder dynamischere Auktions- und Gebotsszenarien. Der Parameter `matchByTaker` spielt hierbei eine Schlüsselrolle, da er bestimmt, ob der Preis des Käufers oder des Verkäufers für die Transaktion verwendet wird.

`matchByTaker` gibt in einem boolischen Wert an, ob der vorgeschlagene Preis des Käufers bzw. Takers verwendet wird.

Die Struktur des Endpunkts, die sowohl `maker`- als auch `taker`-Objekte umfasst, ermöglicht eine detaillierte Spezifikation der Handelsbedingungen. Dies schließt Informationen wie die beteiligten Token, Preise, Gültigkeitszeiträume und spezifische NFT-

Daten ein. Zudem beinhaltet der Endpunkt Mechanismen zur Sicherstellung der Gültigkeit und Authentizität der Handelsaufträge. Es ist wichtig zu beachten, dass die genaue Nutzung und Funktionalität des Endpunkts je nach Implementierung variieren kann. Entwickler haben die Möglichkeit, den Endpunkt an ihre spezifischen Bedürfnisse und Geschäftsmodelle anzupassen.

11.1.6 Fazit

Das Loopring-Protokoll präsentiert sich als eine technisch ausgereifte Lösung für NFT-Marktplätze, die die wesentlichen Funktionalitäten des NFT-Handels – Minten, Verkaufen (Listing) und Kaufen – effizient unterstützt. Die Nutzung der L2-Technologie verspricht dabei erhebliche Vorteile in Bezug auf Transaktionsgeschwindigkeit und -kosten. Trotz dieser vielversprechenden Grundlage zeigt sich jedoch eine deutliche Diskrepanz zwischen dem technischen Potential und der tatsächlichen Adaption in der Entwicklergemeinschaft.
Ein Hauptgrund für diese Diskrepanz liegt in der Komplexität der Entwicklerumgebung. Neueinsteiger sehen sich mit hohem Lernaufwand konfrontiert, der den Einstieg in die Loopring-Infrastruktur erheblich erschwert.
Die Komplexität spiegelt sich auch in der Community wider. Im Discord-Kanal von Loopring finden sich nur wenige Entwickler, die über tiefgreifende Kenntnisse der API und des SDKs verfügen. Dieser Mangel an breiter Expertise innerhalb der Community erschwert den Wissensaustausch und die gegenseitige Unterstützung, die für ein lebendiges Ökosystem entscheidend sind. Die genauen Anforderungen an API-Anfragen, die spezifischen Parameter und insbesondere der EdDSA-Prozess erfordern ein tiefes technisches Verständnis und einen erheblichen Zeitaufwand bei einer Implementierung.
Diese Hürden könnten potenzielle Entwickler und Projekte davon abhalten, Loopring als Basis für ihre NFT-Marktplätze zu wählen, trotz der offensichtlichen Vorteile der Technologie. Die Komplexität der Implementierung steht im Kontrast zu einfacheren, wenn auch möglicherweise weniger leistungsfähigen Alternativen, die eine schnellere Markteinführung ermöglichen.

11.2 Implementierungskonzept auf Taiko

Diese Untersuchung konzentriert sich hauptsächlich auf die Interaktionen mit den konzipierten Smart Contracts mithilfe der Ethers.js-Bibliothek, den Signaturprozess mithilfe der Ethers.js-Bibliothek und die notwendigen Authentifikationsschritten, um den Marktplatz-Smart-Contract als Operator freizugeben.

11.2.1 Konzipierung der Smart Contracts

Die Smart Contracts folgen der Architektur, die in Kapitel 8.4 erläutert wurde. Es wurden drei zentrale Smart Contracts entwickelt: ein Factory-Smart-Contract (A), ein Token-Smart-Contract (B) und ein Marktplatz-Smart-Contract (C), deren Implementierungen im Anhang zu finden sind. Sie bilden den Kern der Implementierung auf Taiko und repräsentieren die fundamentale Logik und Interaktionspunkte des Marktplatzes, vergleichbar mit den vordefinierten Funktionalitäten in der Loopring-Architektur.
Die folgende Analyse konzentriert sich nur auf die notwendigen Interaktionen mit diesen Smart Contracts zur Realisierung der betrachteten User Stories. In diesem Rahmen wird von einer detaillierten Erläuterung des Solidity-Codes abgesehen. Stattdessen werden spezifische Funktionen und deren Arbeitsweise nur dann näher beleuchtet, wenn sie für die Frontend-Interaktion von unmittelbarer Relevanz sind. Die grundlegenden Prinzipien der Funktionalitäten und Datenverwaltung wurden bereits in Kapitel 8.4 allgemein dargelegt und bilden die konzeptuelle Basis für die hier referenzierten Implementierungen.

11.2.2 Minting von NFTs

Für das Minting von NFTs unter Verwendung der bereitgestellten Smart Contracts muss mit zwei Hauptverträgen interagiert werden: Factory.sol (A) und TokenContract.sol (B). Es gibt zwei Szenarien zu betrachten: das Erstellen eines neuen Smart Contracts (bzw. einer Kollektion) und das Minting von Token in einem bestehenden Smart Contract.

Erstellen eines neuen Token-Smart-Contracts
Hierfür wird mit dem Factory-Smart-Contract interagiert. Dabei wird ein neuer Token Contract erstellt und gleichzeitig ein neuer Token gemintet. Der folgende Code zeigt die Interaktionen:

```
1 const ethers = require('ethers');
2 const FactoryABI = require('./path-to-Factory-ABI.json');
3
4 //weiterer Code...
```

Code-Snippet 11.7: JavaScript: Importieren von ethers und der ABI

Die ersten beiden Zeilen des Codes importieren wesentliche Komponenten für die Interaktion mit ethereum-äquivalenten Blockchains. Die Konstante `ethers` lädt die ethers.js-Bibliothek. Im Gegensatz zu den Ausführungen in Kapitel 11.1, wird in den folgenden Code-Beispielen auch der Signaturprozess beleuchtet, der vollständig von der ethers.js-Bibliothek abgedeckt wird.
Die zweite Konstante `FactoryABI` importiert das Application Binary Interface (ABI) des Factory-Smart-Contracts. Das ABI ist eine JSON-Darstellung der Funktionen und

Events des Smart Contracts und ist notwendig für eine korrekte Kommunikation zwischen der JavaScript-Anwendung und dem auf Taiko veröffentlichten Smart Contract. Es definiert die Struktur der Funktionsaufrufe und Events, sodass ethers.js die Daten korrekt formatieren und interpretieren kann.

```
1 const ethers = require('ethers');
2 const FactoryABI = require('./path-to-Factory-ABI.json');
3
4 async function createTokenContractAndMint(
5   _name,
6   _baseURI,
7   _receiver,
8   _amount,
9   _tokenURI,
10  _data) {
11
12 //weiterer Code...
```

Code-Snippet 11.8: JavaScript: createTokenContractAndMint-Funktion I

Die Funktion `createTokenContractAndMint` spiegelt die entsprechende Funktion im Factory-Smart-Contract wider, was die direkte Verbindung zwischen dem JavaScript-Code und der Logik des Smart Contracts unterstreichen soll [A]. Die Funktion akzeptiert genau die sechs Parameter, die den Anforderungen der Factory für die Erstellung eines neuen Token-Smart-Contracts und Token entsprechen.

```
1 //...vorheriger Code
2
3 async function createTokenContractAndMint(...) {
4
5   // Provider fuer Taiko Netzwerk erstellen
6   const provider = new ethers.JsonRpcProvider(
7   "https://rpc.mainnet.taiko.xyz"
8   );
9
10 //weiterer Code...
```

Code-Snippet 11.9: JavaScript: createTokenContractAndMint-Funktion II, provider

Der erste Schritt innerhalb der Funktion ist die Erstellung eines Providers mittels der `ethers` Bibliothek. Dieser Provider dient als Brücke zu Taiko, indem er den Zugriffspunkt “https://rpc.mainnet.taiko.xyz” verwendet. Durch die Einrichtung dieses Providers wird die grundlegende Verbindung zu Taiko hergestellt, sodass mit Smart Contracts interagiert und Transaktionen an das Netzwerk gesendet werden können.

```
1 //...vorheriger Code
2
3 async function createTokenContractAndMint(...) {
```

```
  const provider = new ethers.JsonRpcProvider(
  "https://rpc.mainnet.taiko.xyz");

  const privateKey = 'user_private_key_here';
  // Wallet-Instanz mit privatem Schluessel und Provider
     erstellen
  const wallet = new ethers.Wallet(privateKey, provider);

  const factoryAddress = "0x123456789...";
  // Factory Contract Instanz erstellen
  const factory = new ethers.Contract(
    factoryAddress,
    FactoryABI,
    wallet
  );

//weiterer Code...
```

Code-Snippet 11.10: JavaScript: createTokenContractAndMint-Funktion III, wallet- und factory-Instanz

Der private Schlüssel des Nutzers wird in diesem Beispiel in der Variablen `privateKey` gespeichert. Hier ist wichtig zu betonen, dass dieser Schlüssel nicht auf diese Weise im Code hinterlegt werden sollte; stattdessen sollte er aus einer sicheren, verschlüsselten Umgebung abgerufen werden. Diese Vorgehensweise dient lediglich der Veranschaulichung und dem Verständnis des Codes.
Anschließend wird eine von `ethers` vordefinierte Wallet-Instanz erstellt. Diese Wallet-Instanz kombiniert den privaten Schlüssel mit dem zuvor definierten `provider` und bietet vordefinierte Funktionen zum signieren von Transaktionen [109]. Schließlich wird in der Variable `factory` eine Instanz des Factory-Contracts erstellt, die ebenfalls vordefinierte Funktionen zur Interaktion mit Smart Contracts bietet. Hierbei wird die Adresse des Contracts, das ABI sowie die Wallet-Instanz übergeben [110].

```
//...vorheriger Code

async function createTokenContractAndMint(
  _name,
  _baseURI,
  _receiver,
  _amount,
  _tokenURI
  _data == "0x"
  ) {
  const provider = new ethers.JsonRpcProvider(
  "https://rpc.mainnet.taiko.xyz");
  const privateKey = 'user_private_key_here';
  const wallet = new ethers.Wallet(privateKey, provider);
  const factoryAddress = "0x123456789...";
  const factory = new ethers.Contract(
  factoryAddress, FactoryABI, wallet);

```

```
19   try {
20     // unsignierte Transaktion vorbereiten
21     const unsignedTx = await factory.populateTransaction
22     .createTokenContractAndMint(
23       _name,
24       _baseURI,
25       _receiver,
26       _amount,
27       _tokenURI,
28       _data
29     );
30
31    //weiterer Code...
```

Code-Snippet 11.11: JavaScript: createTokenContractAndMint-Funktion IV

Im `try`-Block beginnt der eigentliche Prozess der Transaktionserstellung und späteren -ausführung. Zunächst wird eine unsignierte Transaktion über eine Methode der `factory`-Instanz vorbereitet. Diese Methode füllt die Transaktion mit den notwendigen Parametern, die am Anfang in die Funktion übergeben wurden, und bereitet eine Transaktion für den Aufruf der `createTokenContractAndMint` Funktion des Factory-Smart-Contracts vor, ohne die Transaktion tatsächlich zu senden [111].

```
1  //...vorheriger Code
2
3    try {
4      const unsignedTx = await factory.populateTransaction
5      .createTokenContractAndMint(...);
6
7      // Transaktion signieren und senden
8      const txResponse = await
          wallet.sendTransaction(unsignedTx);
9      console.log("Transaktion gesendet. Hash:", signedTx.hash);
10
11     // Auf Bestaetigung warten
12     const receipt = await txResponse.wait();
13     console.log("Transaktion bestaetigt in Block:",
14     receipt.blockNumber);
15
16     //weiterer Code...
```

Code-Snippet 11.12: JavaScript: createTokenContractAndMint-Funktion V

Die Transaktion wird über die `wallet`-Instanz signiert und an das Taiko Netzwerk gesendet [112]. Anschließend wartet der Code durch die `.wait()` Methode aus der ethers.js-Bibliothek auf die Bestätigung (`receipt`) der Transaktion [113, 114]. Eine Bestätigung bedeutet, dass die Transaktion vom Taiko-Netzwerk validiert wurde. Sobald die Bestätigung erfolgt ist, wird die Blocknummer, in der die Transaktion inkludiert wurde, ausgegeben.

```
//...vorheriger Code

try {
  const unsignedTx = await factory.populateTransaction
  .createTokenContractAndMint(...);
  const txResponse = await wallet.sendTransaction(unsignedTx);
  console.log("Transaktion gesendet. Hash:", signedTx.hash);
  const receipt = await txResponse.wait();
  console.log("Transaktion bestaetigt in Block:",
     receipt.blockNumber);

//Smart Contract Event aus dem receipt extrahieren
  const event = receipt.logs.map(log => {
      try {
        return factory.interface.parseLog(log);
      } catch (error) {
        return null;
      }
    })
    .find(event => event && event.name ===
       'TokenContractCreated');
  if (!event) {
    throw new Error("TokenContractCreated Event nicht
       gefunden");
  }

  const newTokenContractAddress = event.args.tokenContract;
  console.log("Neuer Token Contract erstellt und Token
     geminted:", newTokenContractAddress);
  return newTokenContractAddress;
} catch (error) {
  console.error("Fehler beim Erstellen des Token Contracts:",
     error);
  throw error;
}
```

Code-Snippet 11.13: JavaScript: createTokenContractAndMint-Funktion VI

Nachdem die Transaktion bestätigt und die Bestätigung vom Taiko Netzwerk empfangen wurde, findet die Verarbeitung der Ereignis-/Eventdaten statt:
Die Extraktion des spezifischen Events “TokenContractCreated” erfolgt durch eine Analyse der Transaktions-Logs. Zunächst werden die Logs aus dem `receipt` extrahiert, die die rohen Daten aller während der Transaktion emittierten Events enthalten. Jedes Log wird dann mit der `interface.parseLog()` Methode des Contract-Interfaces dekodiert, um strukturierte Event-Objekte zu erhalten [115]. Anschließend wird mit der `.find()` Methode das spezifische Event “TokenContractCreated” in der Liste der dekodierten Events gesucht. Sollte das gesuchte Event nicht gefunden werden, wird eine Fehlermeldung geworfen, da dies auf ein unerwartetes Verhalten des Smart Contracts hindeuten würde. Wird das Event erfolgreich lokalisiert, können die relevanten Informationen aus den Event-Logs extrahiert werden [116]. In diesem Fall ist die Aufmerksamkeit auf die Adresse des neu erstellten Token-Contracts gerichtet. Diese Adresse wird

aus dem `args`-Objekt des Events entnommen, speziell aus der Variable `tokenContract`. Abschließend wird diese Adresse in der Konsole ausgegeben, um dem Benutzer oder dem aufrufenden System mitzuteilen, wo der neue Token-Contract zu finden ist. Diese Adresse ist entscheidend für zukünftige Interaktionen mit dem erstellten Token, weitere Minting-Operationen, Transfers oder andere Token-bezogene Aktionen. Die Funktion gibt schließlich diese Adresse zurück, sodass sie für weitere Verarbeitungsschritte oder Benutzerinteraktionen in der aufrufenden Anwendung verwendet werden kann.

Eine vollständige Funktion zum Erstellen eines neuen Token-Smart-Contracts und Minten eines Token auf Taiko könnte im Frontend also folgendermaßen aussehen:

```
const ethers = require('ethers');
const FactoryABI = require('./path-to-Factory-ABI.json');

async function createTokenContractAndMint(
  _name,
  _baseURI,
  _receiver,
  _amount,
  _tokenURI
  _data == "0x") {
  const provider = new ethers.JsonRpcProvider(
  "https://rpc.mainnet.taiko.xyz");
  const privateKey = 'user_private_key_here';
  const wallet = new ethers.Wallet(privateKey, provider);
  const factoryAddress = "0x123456789...";
  const factory = new ethers.Contract(
    factoryAddress,
    FactoryABI,
    wallet);

  try {
    const unsignedTx = await factory.populateTransaction
    .createTokenContractAndMint(
      _name,
      _baseURI,
      _receiver,
      _amount,
      _tokenURI,
      _data);

    const txResponse = await
       wallet.sendTransaction(unsignedTx);
    console.log("Transaktion gesendet. Hash:",
       txResponse.hash);

    const receipt = await txResponse.wait();
    console.log("Transaktion bestaetigt in Block:",
       receipt.blockNumber);

  const event = receipt.logs.map(log => {
      try {
        return factory.interface.parseLog(log);
```

```
40        } catch (error) {
41          return null;
42        }
43      })
44      .find(event => event && event.name ===
          'TokenContractCreated');
45    if (!event) {
46      throw new Error("TokenContractCreated Event nicht
          gefunden");
47    }
48
49    const newTokenContractAddress = event.args.tokenContract;
50    console.log("Neuer Token Contract erstellt und Token
        geminted:", newTokenContractAddress);
51    return newTokenContractAddress;
52
53    } catch (error) {
54      console.error("Fehler beim Erstellen des Listings:",
          error);
55      throw error;
56    }
57  }
```

Code-Snippet 11.14: JavaScript: createTokenContractAndMint-Funktion vollständig

Minten neuer Token in einem bestehenden Token-Smart-Contract

Im Folgenden wird sich auf das Minten zusätzlicher Token innerhalb eines vorhandenen ERC-1155 Token-Smart-Contracts konzentriert.

Der im Anhang B gegebene Token-Smart-Contract bietet eine `mint`-Funktion, die es dem Besitzer ermöglicht, neue Token zu erzeugen. Diese Funktion erwartet Parameter wie die Empfängeradresse der neu erstellten Token, die Menge der zu erstellenden Token, die URI der Metadaten des neuen Token und optional zusätzliche Daten. Um diese Funktionalität im Frontend zu implementieren, kann erneut eine JavaScript-Funktion erstellt werden, die mit dem Token-Smart-Contract und dessen `mint`-Funktion interagiert.

```
1  const TokenContractABI =
     require('./path-to-contract-abi.json');
2
3  async function mintNewToken(
4    _contractAddress,
5    _receiverAddress,
6    _amount,
7    _tokenURI,
8    _data == "0x") {
9    const tokenContract = new ethers.Contract(
10     _contractAddress,
11     TokenContractABI,
12     wallet);
13
14 //weiterer Code...
```

Code-Snippet 11.15: JavaScript: mintNewToken-Funktion I

Wie im vorherigen Code-Beispiel, werden in dieser Implementierung zuerst die ABI des Smart Contracts, mit dem interagiert werden soll, importiert. In diesem Fall ist es die ABI des Token-Smart-Contracts.
Die Funktion `mintNewToken` soll die `mint`-Funktion des Token-Smart-Contracts widerspiegeln und akzeptiert die dafür notwendigen Parameter sowie die Adresse des Smart Contracts, in dem die neuen Token erzeugt werden sollen.
Die Erstellung der Konstanten für einen Provider und eine Wallet-Instanz werden sowohl hier, als auch in den folgenden Beipspielen nicht weiter beleuchtet. Fortlaufend besteht der einzige Unterschied der Funktionen darin, dass mit anderen Contract-Instanz gearbeitet und andere Funktion aufgerufen werden.

Eine vollständige Funktion zum Minten neuer Token in einem bestehenden Smart Contract könnte folgende Struktur aufweisen:

```
1  const TokenContractABI =
       require('./path-to-contract-abi.json');
2
3  async function mintNewToken(
4    _contractAddress,
5    _receiverAddress,
6    _amount,
7    _tokenURI,
8    _data = "0x") {
9    const tokenContract = new ethers.Contract(
10     _contractAddress,
11     TokenContractABI,
12     wallet);
13
14   try {
15     const unsignedTx = await
           tokenContract.populateTransaction.mint(
16       _receiverAddress,
17       _amount,
18       _tokenURI,
19       _data);
20     const txResponse = await
           wallet.sendTransaction(unsignedTx);
21
22     const receipt = await txResponse.wait();
23     console.log("Transaktion best tigt in Block:",
           receipt.blockNumber);
24     const event = receipt.logs.map(...)
25       .find(event => event && event.name === 'TokenMinted');
26
27     if (!event) {
28       throw new Error("TokenMinted Event nicht gefunden");
29     }
```

```
30
31      const tokenId = event.args.tokenId;
32      const amount = event.args.amount;
33      console.log("Neuer Token geminted. TokenId:", tokenId);
34      console.log("Anzahl der neuen Token:", amount);
35
36      return { tokenId, amount };
37    } catch (error) {
38      console.error("Fehler beim Minten:", error);
39      throw error;
40    }
41  }
```

Code-Snippet 11.16: JavaScript: mintNewToken-Funktion vollständig

11.2.3 Verkaufen von NFTs (Erstellen von Listings)

Um Token zu verkaufen, muss mit dem Marktplatz-Smart-Contract interagiert werden, der im Anhang C zu finden ist.
Beim Erstellen von Listings in dem Marktplatz-Smart-Contract muss dieser als erstes als Operator in dem Smart Contract, der den zu verkaufenden Token enthält, vermerkt werden.

Freigabe des Marktplatz-Smart-Contracts als Operator
Diese Freigabe zum Operator wird erreicht, indem mit dem Token-Smart-Contract interagiert und die Standardfunktion `setApprovalForAll()` des ERC-1155 Standards aufgerufen wird [117]. Diese Funktionalität könnte in JavaScript folgendermaßen umgesetzt werden:

```
1  async function approveMarketplaceAsOperator(
2  _tokenContractAddress) {
3    const marketplaceContract = "0x123456789..."
4    const tokenContract = new ethers.Contract(...);
5
6    try {
7      const unsignedTx = await tokenContract.populateTransaction
8      .setApprovalForAll(marketplaceContract, true);
9      const txResponse = await
           wallet.sendTransaction(unsignedTx);
10
11     const receipt = await txResponse.wait();
12     const approvalEvent = receipt.logs.map(...)
13     .find(event => event.event === 'ApprovalForAll');
14
15     if (!approvalEvent) {
16       throw new Error("ApprovalForAll Event nicht gefunden");
17     }
18
19     const { account, operator, approved } =
           approvalEvent.args;
```

```
20      console.log("Marketplace-Freigabe erfolgreich:");
21      console.log("Token-Besitzer:", account);
22      console.log("Marketplace-Adresse:", operator);
23      console.log("Freigabe erteilt:", approved);
24      return {account, operator, approved};
25
26    } catch (error) {
27      console.error(
28      "Fehler bei der Freigabe als Operator:", error);
29      throw error;
30    }
31  }
```

Code-Snippet 11.17: JavaScript: approveMarketplaceAsOperator-Funktion

Die `approveMarketplaceAsOperator`-Funktion spiegelt viele Aspekte wieder, die bereits in vorherigen Abschnitten besprochen wurden. Im Grunde genommen nimmt sie die Adresse des Smart Contracts, der freigegeben werden soll, entgegen und sendet einen Aufruf der `setApprovalForAll`-Funktion in Form einer Transaktion mit den notwendigen Parametern (der Adresse des Marktplatz-Smart-Contracts und ein boolischer Wert) an den Token-Smart-Contract und extrahiert schließlich die Parameter des Events "ApprovalForAll" aus der Bestätigung des Netzwerks. Diese Parameter (`account` als Adresse des Besitzers der Token, `operator` als die Adresse des Marktplatz-Smart-Contract, `approved` als boolischer Wert) werden am Ende zurückgegeben.

Erstellen von Listings im Marktplatz-Smart-Contract

Nachdem der Marktplatz als Operator vermerkt wurde, können Listings für alle Token in dem freigegebenen Token-Smart-Contract erstellt werden. Dazu wird folgendermaßen mit dem Marktplatz interagiert:

```
1  const NFTMarketplaceABI = require(
2  './path-to-NFTMarketplace-abi.json');
3  const marketplaceContract = "0x123456789..."
4  const marketplace = new ethers.Contract(
5      marketplaceContract,
6      NFTMarketplaceABI,
7      wallet);
8
9  async function createListing(
10   _tokenContract,
11   _tokenId,
12   _price,
13   _quantity) {
14
15 //weiterer Code...
```

Code-Snippet 11.18: JavaScript: createListing-Funktion I

Die createListing Funktion spiegelt die Funktion des Smart Contracts wider, um Ver-

kaufsangebote zu erstellen. Dazu nimmt sie die Parameter `tokenContract` und `tokenId` entgegen, um den zu verkaufenden Token zu identifizieren sowie den Preis, für den jeder einzelne Token verkauft werden soll, und die Anzahl der zu verkaufenden Token.

```
//...vorheriger Code

async function createListing(
  _tokenContract,
  _tokenId,
  _price,
  _quantity) {

  try {
  //Umrechnung des _price in Wei
    const priceInWei =
       ethers.utils.parseEther(_price.toString());
    const unsignedTx = await
       marketplace.populateTransaction.createListing(
      _tokenContract,
      _tokenId,
      priceInWei,
      _quantity);

//weiterer Code...
```

Code-Snippet 11.19: JavaScript: createListing-Funktion II

Der Preis des zu verkaufenden Token kann im Frontend in Ether angegeben werden, wird vor dem Erstellen des Listings aber noch in die kleinste Einheit der Blockchain (Wei) umgerechnet, damit das Taiko Netzwerk den Preis richtig vermerkt.

Der Rest dieser Funktion funktioniert genau wie die vorherigen Beispielfunktionen. Die Funktion wird von dem Nutzer signiert, anschließend wird die Bestätigung des Netzwerks gewartet, um daraus das Event des Smart Contracts lesen und die notwendigen Parameter entnehmen zu können.

```
//...vorheriger Code

async function createListing(...) {

  try {
    const priceInWei =
       ethers.utils.parseEther(_price.toString());

    const unsignedTx = await marketplace.populateTransaction
    .createListing(...);
    const txResponse = await
       wallet.sendTransaction(unsignedTx);

    const receipt = await txResponse.wait();
    const event = receipt.logs.map(...).find(
```

```
14          event => event.event === 'ListingCreated'
15        );
16
17        if (!event) {
18          throw new Error("ListingCreated Event nicht gefunden");
19        }
20
21        const listingId = event.args.listingId;
22        console.log("Neues Listing erstellt. ListingId:",
             listingId);
23        return listingId;
24      } catch (error) {
25        console.error("Fehler beim Erstellen des Listings:",
             error);
26        throw error;
27      }
28    }
```

Code-Snippet 11.20: JavaScript: createListing-Funktion III

Durch das Aufrufen dieser Funktion werden die notwendigen Daten an den Smart Contract übermittelt, um ein Verkaufsangebot in der Datensturktur des Marktplatz-Smart-Contracts zu hinterlegen. Der Smart Contract prüft, ob der Absender dieser Funktion Eigentümer der angegebenen Token ist und die angegebene Anzahl besitzt. Schließlich kann dieses Verkaufsangebot im Smart Contract aufgerufen und Token über dieses Listing erworben werden.

Sortieren von Listings über einen Indexer
Im Falle von Taiko, in dem die Marktplatz-Implementierung auf Smart Contracts basiert, können Listings und ihre IDs zwar direkt über Smart-Contract-Funktionen aufgerufen werden, jedoch würde auch hier ein Indexer die Effizienz erheblich steigern. Obwohl alle Informationen theoretisch direkt vom Smart Contract abgerufen werden können, wäre es für eine optimale Benutzererfahrung schneller und effizienter, einen Indexer zu implementieren und in einem zentralen Backend zu hosten.

11.2.4 Kaufen von NFTs

Im Prozess des Kaufens von NFTs ist die Infrastruktur des Marktplatz-Smart-Contracts und die Handhabung der Listing-Strukturen entscheidend. Ein Listing muss im Marktplatz vermerkt sein, damit ein Nutzer eine Kauf-Interaktion damit durchführen kann. Listings werden grundsätzlich anhand ihrer ListingId auf dem Marktplatz identifiziert.

```
1  async function buyNFT(
2    _listingId,
3    _quantity,
4    _totalPrice
5  ) {
```

```
6
7 //weiterer Code...
```

Code-Snippet 11.21: JavaScript: buyNFT-Funktion I

Beim Aufrufen dieser Funktion muss vom Frontend aus die `listingId` und die Anzahl der zu kaufenden Token angegeben werden, da dies die beiden Parameter für die `buyNFT` Funktion des Marktplatz-Smart-Contracts nötig sind, um einen Kauf in die Wege zu leiten. Dazu wird im Frontend berechnet, wieviel Ether der Kauf insgesamt kostet. Dadurch kann der gesamte Betrag, der in der Transaktion als `msg.value` angehangen wird, präzise berechnet werden.

```
1 async function buyNFT(
2   _listingId,
3   _quantity,
4   _totalPrice
5 ) {
6   try {
7     const unsignedTx = await marketplace.populateTransaction
8     .buyNFT(
9       _listingId,
10      _quantity,
11      { value:
            ethers.utils.parseEther(_totalPrice.toString()) }
12    );
13
14 //weiterer Code...
```

Code-Snippet 11.22: JavaScript: buyNFT-Funktion II

In der unsignierten Transaktionsvorbereitung werden die beiden Parameter `listingId` und `quantity` für die Solidity Funktion `buyNFT` übergeben. Diese Funktion ist `payable`, was bedeutet, dass als `msg.value` der Transaktion Ether mitgeschickt werden kann. Dieser `msg.value` Betrag ist in diesem Fall der `totalPrice`, der für die komplette Anzahl der zu erwerbenden Token gezahlt werden muss.

Der Rest der Funktion ähnelt den vorherigen Beispielfunktionen. Eine komplette Funktion für das Kaufen von NFTs auf dem Marktplatz könnte im Frontend folgendermaßen umgesetzt werden:

```
1 async function buyNFT(
2   _listingId,
3   _quantity,
4   _totalPrice
5 ) {
6
7   try {
```

```
8       const unsignedTx = await
           marketplaceContract.populateTransaction.buyNFT(
9         _listingId,
10        _quantity,
11        { value:
             ethers.utils.parseEther(_totalPrice.toString()) }
12      );
13      const txResponse = await
           wallet.sendTransaction(unsignedTx);
14
15      const receipt = await txResponse.wait();
16      const tokenSoldEvent = receipt.logs.map(...).find(
17      event => event.event === 'TokenSold');
18
19      if (!tokenSoldEvent) {
20        throw new Error("TokenSold Event nicht gefunden");
21      }
22
23      const {
24        listingId,
25        tokenContract,
26        tokenId,
27        seller,
28        buyer,
29        quantity,
30        price,
31        mfee
32      } = tokenSoldEvent.args;
33
34      console.log("NFT erfolgreich gekauft:");
35      console.log("Listing ID:", listingId.toString());
36      console.log("Token Contract:", tokenContract);
37      console.log("Token ID:", tokenId.toString());
38      console.log("Verk ufer:", seller);
39      console.log("K ufer:", buyer);
40      console.log("Menge:", quantity.toString());
41      console.log("Preis:", ethers.utils.formatEther(price),
           "ETH");
42      console.log("Marktplatz-Geb hr:",
43      ethers.utils.formatEther(mfee), "ETH");
44
45    } catch (error) {
46      console.error("Fehler beim Kauf des NFTs:", error);
47      throw error;
48    }
49  }
```

Code-Snippet 11.23: JavaScript: buyNFT-Funktion vollständig

11.2.5 Fazit

Die Implementierung eines NFT-Marktplatzes auf Taiko bietet Entwicklern eine vertraute Umgebung, die der Entwicklung auf Ethereum Layer 1 gleicht. Dank der Ethereum-

Äquivalenz von Taiko können Entwickler dieselbe Programmiersprache und dieselben Bibliotheken verwenden, die sie bereits von Ethereum kennen. Alle Tools und Frameworks, die für Ethereum entwickelt wurden, sind auch auf Taiko anwendbar.
Ein wesentlicher Vorteil dieser Äquivalenz ist die große Entwickler-Community, die Taiko unterstützen kann. Da Solidity als Programmiersprache genutzt wird, können Entwickler, die bereits mit Ethereum gearbeitet haben, ohne zusätzlichen Lernaufwand auf Taiko entwickeln. Bibliotheken wie ethers.js bieten umfangreiche Dokumentationen und Unterstützung, was die Entwicklung weiter vereinfacht.

12 Gegenüberstellung: Entwicklerfreundlichkeit

Kriterium	**Loopring**	**Taiko**
Qualität und Vollständigkeit der Dokumentation	Dokumentation vorhanden, teilweise unübersichtlich; wurde in letzter Zeit überarbeitet und weitere Überarbeitungen können folgen	übersichtliche Dokumentation vorhanden
Vertrautheit der Entwicklungsumgebung	proprietäre Umgebung, hoher Lernaufwand, komplexe Struktur	vertraute Umgebung, da ethereum-äquivalent
Verfügbarkeit von Entwicklertools und SDKs	verschiedene Entwicklertools vorhanden, offiziell ein SDK vom Loopring Team	alles was es bereits für Ethereum gibt, kann für Taiko verwendet werden

Tabelle 12.1: Vergleich der Entwicklerfreundlichkeit von Loopring und Taiko

Bewertung

Die Entwicklerfreundlichkeit für Blockchain-Entwickler ist bei Taiko erheblich höher als bei Loopring. Die Entwicklungsumgebung von Loopring erfordert einen beträchtlichen Einarbeitungsaufwand. Die Dokumentation ist teilweise fragmentiert, wodurch es schwierig ist, an einer zentralen Stelle den vollen Kontext für bestimmte Parameter zu finden. Entwickler müssen Informationen aus verschiedenen Quellen zusammentragen, um ein umfassendes Verständnis zu erlangen. Darüber hinaus implementiert Loopring einen anderen Signaturalgorithmus, was bedeutet, dass Entwickler auch in diesem Bereich umfangreiche Kenntnisse erwerben müssen, um eine funktionierende Implementierung zu realisieren.

13 Abschließende Bewertung

In dieser Arbeit wurden die Rollups Loopring und Taiko in den Aspekten Sicherheit, Dezentralisierung, Skalierbarkeit und Entwicklerfreundlichkeit umfassend betrachtet. Beide Skalierungslösungen versuchen, diese Aspekte des Blockchain-Trilemmas zu adressieren, jedoch mit unterschiedlichen Ansätzen und Technologien. Zusammenfassend lassen sich folgende Schlussfolgerungen in den betrachteten Kriterien ziehen:

Dezentralisierung: Loopring ist deutlich früher als Taiko entstanden. In dieser Zeit war eine echte Dezentralisierung für Layer-2-Protokolle noch nicht möglich; stattdessen lag der Fokus auf der Steigerung der Skalierbarkeit, wobei bewusste Kompromisse in der Dezentralisierung eingegangen wurden. Mit seiner fortschrittlichen kryptografischen Implementierung war Loopring seiner Zeit voraus und hat sich als robustes Protokoll etabliert. Im Aspekt der Dezentralisierung hat Taiko einen klaren Vorteil.

Sicherheit: Loopring bietet derzeit eine überlegene Sicherheitsarchitektur im Vergleich zu Taiko, da es für jede Transaktion ZK-Proofs erzeugt. Taiko erzwingt dies aktuell noch nicht auf seinem Rollup und hat erst vor kurzem einen funktionsfähigen zkVerifier in das Protokoll auf dem Mainnet integriert. Zudem könnte das Guardian-Prinzip, das in Taiko implementiert ist, potenziell noch zu Sicherheitskompromissen führen, wenn nicht alle Guardians vertrauenswürdig sind. Taiko adressiert jedoch diese Bedenken bereits und plant, sie in Zukunft zu beseitigen. Hier ist sehr wichtig zu beachten, dass Taiko sich noch in einem frühen Entwicklungsstadium befindet.

Skalierbarkeit: In Bezug auf die Skalierbarkeit hat Loopring aufgrund seiner zentralisierten Architektur einen klaren Vorteil. Zentralisierte Systeme können schnellere Verarbeitungsgeschwindigkeiten bieten, was für Anwendungen mit hohem Transaktionsvolumen entscheidend ist. Taiko hingegen hat großes Potenzial für die Zukunft, insbesondere durch das innovative Inception-Layer-Prinzip. Während Taiko möglicherweise in Zukunft durch diverse Rollup-Schichten ähnliche Transaktionskapazitäten erreichen kann, bleibt Loopring aufgrund seiner aktuellen Implementierung führend in dieser Kategorie.

Entwicklerfreundlichkeit: Bei der Entwicklerfreundlichkeit schneidet Taiko besser ab. Entwickler können auf eine breite Community zurückgreifen und sind nicht auf das Feedback eines spezifischen Teams angewiesen, um eine vollständig funktionierende Implementierung zu erreichen. Dies erleichtert den Einstieg und die Entwicklung von Anwendungen erheblich. Im Gegensatz dazu müssen Entwickler bei Loopring oft auf die Unterstützung des Loopring-Teams zurückgreifen, um sich im Ökosystem zurechtzufinden. Wenn Loopring keine ausreichenden Ressourcen zur Verfügung stellt, könnte dies den Einarbeitungsprozess erheblich verlängern.

Beide Protokolle sind gut geeignet, um einen NFT-Marktplatz zu implementieren und reflektieren die moderne Ausrichtung auf ZK Architekturen im Ethereum-Ökosystem.

Loopring ist eine geeignete Wahl, wenn der Fokus auf einer einfachen Handelsplattform liegt, die kostengünstiges Minting, schnelle Transaktionsbestätigungen und hohe Sicherheit priorisiert. Dieser Rollup zeichnet sich durch eine robuste Sicherheitsarchitektur aus, die sie besonders attraktiv für Projekte macht, bei denen Kosteneffizienz, Geschwindigkeit und Vertrauen im Vordergrund stehen.
Es ist jedoch wichtig zu berücksichtigen, dass Loopring eine zentralisierte Lösung ist. Der Betrieb und die Funktionalität der Plattform hängen in erheblichem Maße von dem zentralen Relayer ab, der für die Verarbeitung der Transaktionen verantwortlich ist. Diese Abhängigkeit kann die Autonomie des Systems einschränken und erfordert eine sorgfältige Abwägung, ob diese zentralisierte Struktur mit den Zielen des jeweiligen Projekts vereinbar ist.
Darüber hinaus ist es oft notwendig, die Unterstützung des Loopring-Teams in Anspruch zu nehmen, um Entwicklern den Einstieg in das Ökosystem zu erleichtern und sicherzustellen, dass sie die vorhandenen Werkzeuge und Schnittstellen effektiv nutzen können.

Taiko hingegen stellt eine hervorragende Erweiterung der Ethereum-Blockchain dar, bei der Dezentralität und Robustheit im Mittelpunkt stehen. Durch seine Ethereum-Äquivalenz ermöglicht es Taiko, maßgeschneiderte NFT-Marktplätze mit einer Vielzahl von Funktionen zu erstellen. Diese Flexibilität macht Taiko besonders attraktiv, wenn komplexe Verkaufsmodelle wie Auktionen, Belohnungssysteme für Käufer oder Auszahlungen an mehrere Verkäufer implementiert werden sollen.
Die Plattform bietet Entwicklern die Freiheit, innovative Lösungen zu entwerfen, die sich an die spezifischen Bedürfnisse ihrer Nutzer anpassen. Ein weiterer entscheidender Vorteil von Taiko ist, dass die Entwicklung unabhängig von der Unterstützung eines spezifischen Teams erfolgen kann. Stattdessen steht die gesamte Ethereum-Community als Ressource zur Verfügung, um Fragen zu beantworten und Herausforderungen zu meistern.
Es ist jedoch zu beachten, dass das Minting auf Taiko aufgrund der Nähe zur Ethereum-Blockchain mit höheren Kosten verbunden ist. Zudem sind längere Bestätigungszeiten ein Faktor, der bei der Planung berücksichtigt werden sollte. Diese entstehen durch den Fokus auf Sicherheit und Dezentralität, die essenzielle Werte des Rollups darstellen. Für Projekte, die auf robuste und flexible Lösungen setzen, können diese Kompromisse jedoch gerechtfertigt sein.

Anhang A: Factory.sol

https://github.com/codingdani/JamzrSC/blob/main/Factory.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "./TokenContract.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract CustomERC1155Factory is Ownable {
    event TokenContractCreated(address indexed creator,
       address indexed tokenContract, string name);

    address[] public createdContracts;
    mapping(address => address[]) public creatorToContracts;

    constructor(address initialOwner) Ownable(initialOwner) {}

    function createTokenContract(string memory _baseURI,
       string memory _name) public
    returns (address) {
        TokenContract newContract = new TokenContract(_name,
           _baseURI, msg.sender);
        createdContracts.push(address(newContract));
        creatorToContracts[msg.sender].push(address(
        newContract));
        emit TokenContractCreated(msg.sender,
           address(newContract), _name);
        return address(newContract);
    }

    function createTokenContractAndMint(
        string memory _name,
        string memory _baseURI,
        address _receiver,
        uint256 _amount,
        string memory _tokenURI,
        bytes memory _data
    ) public returns (address, uint256) {
        TokenContract newContract = new TokenContract(_name,
           _baseURI, address(this));

        // Mint tokens
        uint256 tokenId = newContract.mint(
            _receiver,
            _amount,
            _tokenURI,
            _data);

        // Add contract to tracking arrays and mappings
        createdContracts.push(address(newContract));

```

```
45          // Transfer ownership to the caller (msg.sender)
46          newContract.transferOwnership(msg.sender);
47          creatorToContracts[msg.sender].push(address(
48          newContract));
49
50          // Emit event
51          emit TokenContractCreated(msg.sender,
               address(newContract), _name);
52          return (address(newContract), tokenId);
53      }
54
55      function getAllContractsByCreator(address _creator)
           public view
56      returns (address[] memory) {
57          return creatorToContracts[_creator];
58      }
59
60  }
```

Anhang B: TokenContract.sol

https://github.com/codingdani/JamzrSC/blob/main/TokenContract.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC1155/ERC1155.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/Strings.sol";

contract TokenContract is ERC1155, Ownable {
    using Strings for uint256;

    event TokenMinted(uint256 indexed tokenId, address
       indexed receiver, uint256 amount, string tokenURI);

    string public name;
    uint256 private nextTokenId;

    struct TokenDetails {
    uint256 totalSupply;
    string uri;
    }
    //maps token ID to TokenDetails
    mapping(uint256 => TokenDetails) private tokenDetails;

    constructor(string memory _name, string memory _baseURI,
       address _owner)
    ERC1155(_baseURI)
    Ownable(_owner) {
        name = _name;
    }

    function mint(
        address _receiver,
        uint256 _amount,
        string calldata _tokenURI,
        bytes calldata _data
    ) external onlyOwner returns (uint256) {
        require(_amount > 0, "Amount must be greater than 0");
        uint256 tokenId = nextTokenId++;
        tokenDetails[tokenId] = TokenDetails({
            totalSupply: _amount,
            uri: _tokenURI,
        });
        _mint(_receiver, tokenId, _amount, _data);
        emit TokenMinted(tokenId, _receiver, _amount,
           _tokenURI);
        return tokenId;
    }

    function mintBatch(
```

```
47          address _receiver,
48          uint256[] calldata _amounts,
49          string[] calldata _uris,
50          bytes calldata _data
51      ) external onlyOwner {
52          require(_amounts.length == _uris.length,
53                  "Arrays length mismatch");
54          uint256[] memory newIds = new
               uint256[](_amounts.length);
55          for (uint256 i = 0; i < _amounts.length; i++) {
56              require(_amounts[i] > 0, "Amount must be greater
                   than 0");
57              uint256 newId = nextTokenId++;
58              newIds[i] = newId;
59              tokenDetails[newId] = TokenDetails({
60                  totalSupply: _amounts[i],
61                  uri: _uris[i],
62              });
63              emit TokenMinted(newId, _receiver, _amounts[i],
                   _uris[i]);
64          }
65          _mintBatch(_receiver, newIds, _amounts, _data);
66      }
67
68      function getTokenDetails(uint256 _tokenId) external view
           returns (
69          uint256 totalSupply,
70          string memory tokenUri
71          ) {
72      TokenDetails memory details = tokenDetails[_tokenId];
73      return (details.totalSupply, details.uri);
74      }
75
76      function setTokenURI(uint256 _tokenId, string memory
           _newURI) external onlyOwner {
77          tokenDetails[_tokenId].uri = _newURI;
78      }
79
80      function uri(uint256 _tokenId) public view virtual
           override
81      returns (string memory) {
82          string memory tokenURI = tokenDetails[_tokenId].uri;
83          if (bytes(tokenURI).length > 0) {
84              return tokenURI;
85          }
86      }
87
88      function setName(string memory _newName) external
           onlyOwner {
89          name = _newName;
90      }
91  }
```

Anhang C: Marketplace.sol

https://github.com/codingdani/JamzrSC/blob/main/Marketplace.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC1155/IERC1155.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import
   "@openzeppelin/contracts/utils/structs/EnumerableSet.sol";

contract NFTMarketplace is ReentrancyGuard, Ownable {
    using EnumerableSet for EnumerableSet.UintSet;

    struct Listing {
        uint256 listingId;
        address tokenContract;
        address seller;
        uint256 tokenId;
        uint256 price;
        uint256 quantity;
        bool isActive;
        uint256 timestamp;
    }
    //maps listindId to Listing
    mapping(uint256 => Listing) public listings;
    uint256 private listingCounter;

    EnumerableSet.UintSet internal activeListings;
    //maps sellerAddress to Set of listingId
    mapping(address => EnumerableSet.UintSet) internal
       sellerListings;
    //maps tokenContractAddress to TokenId to Set of listingId
    mapping(address => mapping(uint256 =>
       EnumerableSet.UintSet)) internal assignTokenToListings;
    //maps listingId to bool to indicate that Listing needs
       to be removed from all mappings and Sets
    mapping(uint256 => bool) public listingNeedsCleanup;

    mapping(address => bool) public supportedTokenContracts;
    uint256 public marketplaceFeePercentage;

    event ListingCreated(
        uint256 indexed listingId,
        address tokenContract,
        address indexed seller,
        uint256 tokenId,
        uint256 price,
        uint256 quantity,
        uint createdAt
        );
    event ListingDeactivated(uint256 indexed listingId);
```

```
47      event ListingDeleted(uint256 indexed listingId);
48      event ListingUpdated(uint256 indexed listingId, uint256
           remainingQuantity);
49      event NFTPurchased(uint256 indexed listingId, address
           indexed buyer, uint256 quantity);
50      event TokenSold(
51          uint256 indexed listingId,
52          address indexed tokenContract,
53          address indexed seller,
54          address buyer,
55          uint256 tokenId,
56          uint256 quantity,
57          uint256 price,
58          uint256 mfee,
59          uint256 timestamp
60          );
61      event MarketplaceFeeUpdated(uint256 newFeePercentage);
62
63      constructor(uint256 _initialFeePercentage)
           Ownable(msg.sender){
64          marketplaceFeePercentage = _initialFeePercentage;
65      }
66
67  //All Listing Operations: create, delete, removeFromMappings,
       deactivate, cleanup
68  // Listing-Utils: getAllActiveListings, getListingsForToken,
       getSellerListings, getListingDetailsFromId
69
70      function createListing(
71          address _tokenContract,
72          uint256 _tokenId,
73          uint256 _price,
74          uint256 _quantity)
75      external returns (uint256) {
76          require(supportedTokenContracts[_tokenContract],
77          "Token contract not supported");
78          require(IERC1155(_tokenContract).balanceOf(
79              msg.sender, _tokenId) >= _quantity,
80          "Insufficient token balance");
81      //marketplace has to be set as Operator in
           Token-Smart-Contract
82          require(IERC1155(_tokenContract).isApprovedForAll(
83          msg.sender, address(this)),
84          "Contract not approved as operator");
85          uint256 listingId = _getNextListingId();
86          Listing memory newListing = Listing(
87              listingId,
88              _tokenContract,
89              msg.sender,
90              _tokenId,
91              _price,
92              _quantity,
93              true,
94              block.timestamp
95              );
96          listings[listingId] = newListing;
```

```
97          sellerListings[msg.sender].add(listingId);
98          activeListings.add(listingId);
99          assignTokenToListings[_tokenContract][_tokenId]
100         .add(listingId);
101         emit ListingCreated(
102             newListing.listingId,
103             newListing.tokenContract,
104             newListing.seller,
105             newListing.tokenId,
106             newListing.price,
107             newListing.quantity,
108             newListing.timestamp
109             );
110         return listingId;
111     }
112
113     function deleteListing(uint256 _listingId) external {
114         Listing storage listing = listings[_listingId];
115         require(listing.seller == msg.sender, "Not the
               seller");
116         activeListings.remove(_listingId);
117         _removeFromSellerListings(listing.seller, _listingId);
118         _removeFromAssignTokenToListings(
119         listing.tokenContract, listing.tokenId, _listingId);
120         delete listingNeedsCleanup[_listingId];
121         delete listings[_listingId];
122         emit ListingDeleted(_listingId);
123     }
124
125     function _removeFromSellerListings(address _seller,
           uint256 _listingId) internal {
126         sellerListings[_seller].remove(_listingId);
127     }
128
129     function _removeFromAssignTokenToListings(address
           _tokenContract, uint256 _tokenId, uint256 _listingId)
           internal {
130         assignTokenToListings[_tokenContract][_tokenId]
131         .remove(_listingId);
132     }
133
134     function _deactivateListing(Listing storage _listing,
           uint256 _listingId) internal {
135         require(_listing.isActive, "Listing already
               deactivated");
136         _listing.isActive = false;
137         activeListings.remove(_listingId);
138         listingNeedsCleanup[_listingId] = true;
139         emit ListingDeactivated(_listingId);
140     }
141
142     function cleanupListing(uint256 _listingId) external {
143     //should be called by listing.seller or a reward could be
           included for all Users to collect after cleaning
144         require(listingNeedsCleanup[_listingId], "Listing
               doesn't need cleanup");
```

```
145         Listing storage listing = listings[_listingId];
146         _removeFromSellerListings(listing.seller, _listingId);
147         _removeFromAssignTokenToListings(
148         listing.tokenContract, listing.tokenId, _listingId);
149         delete listingNeedsCleanup[_listingId];
150         delete listings[_listingId];
151         emit ListingDeleted(_listingId);
152     }
153
154 //Listing Utils
155     function _getNextListingId() private returns (uint256) {
156         listingCounter++;
157         return listingCounter;
158     }
159
160     function getAllActiveListings(uint256 _startIndex,
        uint256 _count)
161     external view returns (Listing[] memory, bool) {
162     //This Function can load a specific number of Active
        Listings, which can be called from the Frontend
163         uint256 totalActive = activeListings.length();
164         uint256 endIndex = _startIndex + _count;
165         if (endIndex > totalActive) {
166             endIndex = totalActive;
167         }
168         uint256 resultCount = endIndex - _startIndex;
169         Listing[] memory result = new Listing[](resultCount);
170         for (uint256 i = 0; i < resultCount; i++) {
171             uint256 listingId = activeListings.at(_startIndex
                + i);
172             result[i] = listings[listingId];
173         }
174     //gibt das Array von Listings zur ck und einen boolean,
        ob noch mehr Listings existieren
175         return (result, endIndex < totalActive);
176     }
177
178     function getListingsForToken(address _tokenContract,
        uint256 _tokenId)
179     external view returns (uint256[] memory) {
180         return assignTokenToListings[_tokenContract][_tokenId]
181         .values();
182     }
183
184     function getSellerListings(address _seller) external view
        returns (uint256[] memory) {
185         return sellerListings[_seller].values();
186     }
187
188     function getListingDetailsFromId(uint256 _listingId)
189     external view returns (Listing memory) {
190         require(_listingId <= listingCounter, "Listing does
            not exist");
191         return listings[_listingId];
192     }
193
```

```
194
195 //Purchase NFT Function
196 // no state updating (mappings and arrays) so the last buyer
    of the Token does not get punished with high gas fees when
    the Listing should be deactivated
197
198     function buyNFT(
199         uint256 _listingId,
200         uint256 _quantity)
201     external payable nonReentrant {
202         Listing storage listing = listings[_listingId];
203         require(listing.isActive, "Listing not active");
204         require(_quantity <= listing.quantity, "Insufficient
               quantity available");
205         uint256 totalPrice = listing.price * _quantity;
206         require(msg.value >= totalPrice, "Insufficient
               payment");
207         uint256 fee = (totalPrice * marketplaceFeePercentage)
               / 10000;
208         uint256 sellerPayment = totalPrice - fee;
209
210     //update quantity
211         listing.quantity -= _quantity;
212         if (listing.quantity == 0) {
213             _deactivateListing(listing, _listingId);
214             emit ListingDeactivated(_listingId);
215         }
216
217     // Perform transfers
218         (bool success, ) =
               payable(listing.seller).call{value:
               sellerPayment}("");
219         require(success, "Transfer to seller failed");
220         (success, ) = payable(owner()).call{value: fee}("");
221         require(success, "Transfer of fee failed");
222         IERC1155(listing.tokenContract)
223         .safeTransferFrom(address(this), msg.sender,
               listing.tokenId, _quantity, "");
224
225     // record transaction & emit TokenSold event
226     emit TokenSold(
227         _listingId,
228         listing.tokenContract,
229         listing.seller,
230         msg.sender,
231         listing.tokenId,
232         _quantity,
233         totalPrice,
234         fee,
235         block.timestamp
236     );
237     emit ListingUpdated(_listingId, listing.quantity);
238     }
239
240
241 //Marketplace Functions
```

```
242     function setMarketplaceFee(uint256 _newFeePercentage)
           external onlyOwner {
243         require(_newFeePercentage <= 10000, "Fee percentage
               cannot exceed 100%");
244     //fee shown is 100.00% a 5 percent fee would be 500//
245         marketplaceFeePercentage = _newFeePercentage;
246         emit MarketplaceFeeUpdated(_newFeePercentage);
247     }
248 
249     function addSupportedTokenContract(address
           _tokenContract) external onlyOwner {
250         supportedTokenContracts[_tokenContract] = true;
251     }
252 
253     function removeSupportedTokenContract(address
           _tokenContract) external onlyOwner {
254         supportedTokenContracts[_tokenContract] = false;
255     }
256 }
```

Literatur

[1] etherscan. *Recorded Gas Prices on Ethereum*. URL: https://etherscan.io/chart/gasprice (aufgerufen am 23. 12. 2024).

[2] etherscan.io. *Ethereum Average Block Time Chart | Etherscan*. URL: https://etherscan.io/chart/blocktime (aufgerufen am 06. 12. 2024).

[3] etherscan.io. *Ethereum Daily Transactions Chart | Etherscan*. URL: https://etherscan.io/chart/tx (aufgerufen am 06. 12. 2024).

[4] Ethereum Foundation. *The Merge*. URL: https://ethereum.org/en/roadmap/merge/ (aufgerufen am 06. 12. 2024).

[5] VB et al. *EIP-1559: Fee market change for ETH 1.0 chain*. URL: https://eips.ethereum.org/EIPS/eip-1559 (aufgerufen am 06. 12. 2024).

[6] Anonymous. (2022) *Ethereum 2.0 Roadmap - Die technische Implementierung*. URL: https://blockchainwelt.de/ethereum-2-0/roadmap/ (aufgerufen am 06. 12. 2024).

[7] Ethereum Foundation. *Danksharding*. URL: https://ethereum.org/en/roadmap/danksharding/ (aufgerufen am 06. 12. 2024).

[8] A Jain, Y Punjabi und R Mathur (2024) EXPLORING THE EFFICACY OF ROLLUPS' A COMPARATIVE STUDY OF OPTIMISTIC AND ZKROLLUPS AND THEIR POPULAR IMPLEMENTATIONS. *Journal of Analysis and Computations* XVIII(1):S. 297–315.

[9] Ethereum Foundation. *Sidechains*. URL: https://ethereum.org/en/developers/docs/scaling/sidechains/ (aufgerufen am 06. 12. 2024).

[10] Ethereum Foundation. *Validium*. URL: https://ethereum.org/en/developers/docs/scaling/validium/ (aufgerufen am 06. 12. 2024).

[11] Ethereum Foundation. *State Channels*. URL: https://ethereum.org/en/developers/docs/scaling/state-channels/ (aufgerufen am 06. 12. 2024).

[12] Ethereum Foundation. *Plasma chains*. URL: https://ethereum.org/en/developers/docs/scaling/plasma/ (aufgerufen am 06. 12. 2024).

[13] Ethereum Foundation. *Volitions and Validium*. URL: https://ethereum.org/en/developers/docs/scaling/validium/#volitions-and-validium (aufgerufen am 06. 12. 2024).

[14] Ethereum Foundation. *Optimistic Rollups Chapter: Censorship Resistance*. URL: https://ethereum.org/en/developers/docs/scaling/optimistic-rollups/#censorship-resistance (aufgerufen am 06. 12. 2024).

[15] Ethereum Foundation. *Optimistic Rollups Chapter: How do Optimistic Rollups interact with Ethereum?* URL: https://ethereum.org/en/developers/docs/scaling/optimistic-rollups/#optimistic-rollups-and-ethereum (aufgerufen am 06. 12. 2024).

[16] Ethereum Foundation. *Optimistic Rollups Chapter: What is an Optimistic Rollup?* URL: https://ethereum.org/en/developers/docs/scaling/optimistic-rollups/#what-is-an-optimistic-rollup (aufgerufen am 06. 12. 2024).

[17] R David. *Layer2 Proof Series*. URL: https://www.bnbchain.org/en/blog/layer2-proof-series-part-1-op-stack-fraud-proof-problems-solutions-and-innovations (aufgerufen am 16. 12. 2024).

[18] Alchemy. *How Do Optimistic Rollups Work (The Complete Guide)*. URL: https://www.alchemy.com/overviews/optimistic-rollups#how-do-fraud-proofs-work (aufgerufen am 06. 12. 2024).

[19] Ethereum Foundation. *Optimistic Rollups Chapter: EVM Compatibility*. URL: https://ethereum.org/en/developers/docs/scaling/optimistic-rollups/#evm-compatibility (aufgerufen am 06. 12. 2024).

[20] Ethereum Foundation. *Optimistic Rollups Chapter: Pros and Cons*. URL: https://ethereum.org/en/developers/docs/scaling/optimistic-rollups/#optimistic-rollups-pros-and-cons (aufgerufen am 06. 12. 2024).

[21] Optimism Foundation. URL: https://www.optimism.io/ (aufgerufen am 16. 12. 2024).

[22] Arbitrum. URL: https://arbitrum.io/ (aufgerufen am 16. 12. 2024).

[23] Vitalik Buterin. *The different types of ZK-EVMs*. URL: https://vitalik.eth.limo/general/2022/08/04/zkevm.html (aufgerufen am 06. 12. 2024).

[24] Anonymous. *Was ist Loopring und wie funktioniert es? Wer hat LRC kreiert?* URL: https://kriptomat.io/de/kryptowaehrungen-kurse/loopring-lrc-kurs/was-ist/ (aufgerufen am 06. 12. 2024).

[25] Anonymous. *Loopring - Decentralized Finance | IQ.wiki*. URL: https://iq.wiki/wiki/loopring (aufgerufen am 06. 12. 2024).

[26] LoopringTeam. *Loopring - zkRollup Layer 2 for Trading and Payment*. URL: https://loopring.org/#/about (aufgerufen am 06. 12. 2024).

[27] L2Beats. *Loopring Smart Contracts*. URL: https://l2beat.com/scaling/projects/loopring#contracts (aufgerufen am 06. 12. 2024).

[28] LoopringTeam. *DefaultDepositContract.sol at master · Loopring/protocols*. URL: https://github.com/Loopring/protocols/blob/master/packages/loopring_v3/contracts/core/impl/DefaultDepositContract.sol (aufgerufen am 06. 12. 2024).

[29] LoopringTeam. *BlockVerifier.sol at master · Loopring/protocols*. URL: https://github.com/Loopring/protocols/blob/master/packages/loopring_v3/contracts/core/impl/BlockVerifier.sol (aufgerufen am 06. 12. 2024).

[30] Least Authority. *Security Audit for Loopring*. URL: https://leastauthority.com/static/publications/LeastAuthority_Loopring_3.6_Design_Implementation_Smart_Contracts_Final_Audit_Report.pdf (aufgerufen am 06. 12. 2024).

[31] LoopringTeam. *Loopring Design Document*. URL: https://github.com/Loopring/protocols/blob/master/packages/loopring_v3/DESIGN.md (aufgerufen am 06. 12. 2024).

[32] LoopringTeam. *Loopring Design Doc - Introduction*. URL: https://github.com/Loopring/protocols/blob/master/packages/loopring_v3/DESIGN.md#introduction (aufgerufen am 09. 12. 2024).

[33] Ethereum Foundation. *Zero-Knowledge Rollups: Validity Proofs*. URL: https://ethereum.org/en/developers/docs/scaling/zk-rollups/#validity-proofs (aufgerufen am 06. 12. 2024).

[34] C Fairy. *Trusted Setup in zkSNARKs*. URL: https://medium.com/coinmonks/trusted-setup-in-zksnarks-powers-of-tau-vs-lagrange-basis-7f12978f1eb9 (aufgerufen am 16. 12. 2024).

[35] Ethereum Foundation. *Zero-Knowledge Rollups: Censorship Resistance*. URL: https://ethereum.org/en/developers/docs/scaling/zk-rollups/#censorship-resistance (aufgerufen am 06. 12. 2024).

[36] D Wang, J Zhou, M Finestone und A Wang Loopring: A Decentralized Token Exchange Protocol.

[37] LoopringTeam. (2023) *LRC Staking | Loopring Smart Wallet*. URL: https://docs-wallet.loopring.io/earn/staking/lrc-staking (aufgerufen am 06. 12. 2024).

[38] M Finestone. (2020) *LRC Staking and Claiming Instructions*. URL: https://medium.com/loopring-protocol/lrc-staking-and-claiming-instructions-91fd80e1af98 (aufgerufen am 06. 12. 2024).

[39] LoopringTeam. *Loopring Design Document: Exchange Staking*. URL: https://github.com/Loopring/protocols/blob/master/packages/loopring_v3/DESIGN.md#exchange-staking (aufgerufen am 06. 12. 2024).

[40] M Finestone. (2022) *LRC Tokenomics v2*. URL: https://medium.com/loopring-protocol/lrc-tokenomics-v2-1e6fd99e9e9c (aufgerufen am 06. 12. 2024).

[41] Anonymous. *Loopring Proposals*. URL: https://vote.loopring.io/#/ (aufgerufen am 06. 12. 2024).

[42] LoopringTeam. (2024) *Loopring on X: Community Call*. URL: https://x.com/loopringorg/status/1752346447100498079 (aufgerufen am 06. 12. 2024).

[43] LoopringTeam. (2022) *API References | Loopring Dev Docs*. URL: https://docs-protocol.loopring.io/counterfactual-nft/api-references (aufgerufen am 09. 12. 2024).

[44] taikoscan.io. *Taiko Mainnet Blocks | Taiko Mainnet*. URL: https://taikoscan.io/blocks (aufgerufen am 06. 12. 2024).

[45] TaikoLabs. *Introduction to Taiko*. URL: https://taiko.mirror.xyz/oRy3ZZ_4-6IEQcuLCMMlxvdH6E-T3_H7UwYVzGDsgf4 (aufgerufen am 06. 12. 2024).

[46] D Wang. *Based Contestable Rollup (BCR)*. URL: https://taiko.mirror.xyz/Z4I5ZhreGkyfdaL5I9P0Rj0DNX4zaWFmcws-0CVMJ2A (aufgerufen am 06. 12. 2024).

[47] TaikoLabs. *Taiko Whitepaper*. URL: https://github.com/code-423n4/2024-03-taiko/blob/main/packages/protocol/docs/taiko-whitepaper.pdf (aufgerufen am 10. 12. 2024).

[48] TaikoLabs. *Taiko nodes*. URL: https://docs.taiko.xyz/taiko-protocol/taiko-nodes/ (aufgerufen am 10. 12. 2024).

[49] TaikoLabs. *Differences from Ethereum*. URL: https://docs.taiko.xyz/network-reference/differences-from-ethereum/ (aufgerufen am 06. 12. 2024).

[50] *taikoxyz/raiko*. original-date: 2023-09-07T15:35:46Z. 5. Dez. 2024. URL: https://github.com/taikoxyz/raiko (aufgerufen am 06. 12. 2024).

[51] *succinctlabs/sp1*. original-date: 2023-12-05T00:54:44Z. 6. Dez. 2024. URL: https://github.com/succinctlabs/sp1 (aufgerufen am 06. 12. 2024).

[52] Anonymous. *RISC Zero*. URL: https://risczero.com (aufgerufen am 06. 12. 2024).

[53] Intel. *Intel® Software Guard Extensions (Intel®SGX)*. URL: https://www.intel.de/content/www/de/de/products/docs/accelerator-engines/software-guard-extensions.html (aufgerufen am 06. 12. 2024).

[54] Gate.io. (2024) *Entwicklung der TEE-Technologie und Anwendungen*. URL: https://www.gate.io/de/learn/articles/evolution-of-tee-technology-and-applications/3830 (aufgerufen am 09. 12. 2024).

[55] TaikoLabs. *Based Rollup Economics*. URL: https://taiko.mirror.xyz/PhlvGdIaY3-ZQ1DqI9uM5LxrWGWLAzLI84rkxhvPKmM (aufgerufen am 06. 12. 2024).

[56] TaikoLabs. *Multi-proofs*. URL: https://docs.taiko.xyz/core-concepts/multi-proofs/ (aufgerufen am 06. 12. 2024).

[57] L2BEATS. *L2Beat - Taikos Smart Contracts*. URL: https://l2beat.com/scaling/projects/taiko#contracts (aufgerufen am 09. 12. 2024).

[58] OpenZeppelin Security. *Taiko Protocol Audit*. URL: https://blog.openzeppelin.com/taiko-protocol-audit (aufgerufen am 10. 12. 2024).

[59] TaikoLabs. *Terminology*. URL: https://docs.taiko.xyz/resources/terminology/ (aufgerufen am 09. 12. 2024).

[60] TaikoLabs. *Enable a prover*. URL: https://docs.taiko.xyz/guides/node-operators/enable-a-prover/ (aufgerufen am 09. 12. 2024).

[61] TaikoLabs. *Contestable rollups*. URL: https://docs.taiko.xyz/core-concepts/contestable-rollup/ (aufgerufen am 06. 12. 2024).

[62] TaikoLabs. *Taiko's Approach to Multi-Proofs — Taiko Labs*. URL: https://taiko.mirror.xyz/j_zUGgLDwb1FY18fzh7bJQz2Qt5xbUlqov4n-vm6IC0 (aufgerufen am 06. 12. 2024).

[63] TaikoLabs. *Taiko Secures Its Based Rollup with ZK Proofs*. URL: https://taiko.mirror.xyz/4c6VNhjKLHOMaNKRryyKHkiHcWx7caRax_mC0jTr-sY (aufgerufen am 06. 12. 2024).

[64] TaikoLabs. *Zeroing into zkVMs*. URL: https://taiko.mirror.xyz/e_5GeGGFJIrOxqvXOfzY6 (aufgerufen am 10. 12. 2024).

[65] TaikoLabs. *What is Taiko*. URL: https://docs.taiko.xyz/core-concepts/what-is-taiko/ (aufgerufen am 10. 12. 2024).

[66] *taikoxyz/dao-contracts*. original-date: 2024-05-07T12:08:39Z. 9. Nov. 2024. URL: https://github.com/taikoxyz/dao-contracts (aufgerufen am 09. 12. 2024).

[67] Anonymous. *Taiko Tokenomics: Total Supply of 1 Billion Token*. URL: https://www.bitget.com/news/detail/12560604014108 (aufgerufen am 09. 12. 2024).

[68] LoopringTeam. *Loopring Design Document - Merkle Tree*. URL: https://github.com/Loopring/protocols/blob/master/packages/loopring_v3/DESIGN.md#merkle-tree (aufgerufen am 12. 12. 2024).

[69] M Finestone. (2021) *How to start a Loopring relayer node?* URL: www.reddit.com/r/loopringorg/comments/m12iza/how_to_start_a_loopring_relayer_node/gqc66pt/ (aufgerufen am 09. 12. 2024).

[70] LoopringTeam. (2022) *WebSocket | Loopring Dev Docs*. URL: https://docs-protocol.loopring.io/resources/websocket (aufgerufen am 09. 12. 2024).

[71] LoopringTeam. (2022) *Signature | Loopring Dev Docs*. URL: https://docs-protocol.loopring.io/resources/signature (aufgerufen am 09. 12. 2024).

[72] TaikoLabs. *Taiko Protocol Overview*. URL: https://taiko.mirror.xyz/y_47kIOL5kavvBmG0zVujD2TRztMZt-xgM5d4oqp4_Y (aufgerufen am 11. 12. 2024).

[73] TaikoLabs. *RPC configuration*. URL: https://docs.taiko.xyz/network-reference/rpc-configuration/ (aufgerufen am 09. 12. 2024).

[74] Byron. (2024) *Loopring Quarterly Update (Q3/2024)*. URL: https://medium.com/loopring-protocol/loopring-quarterly-update-q3-2024-8bbdf5eb78eb (aufgerufen am 09. 12. 2024).

[75] L2BEAT. *L2BEAT Governance*. URL: https://l2beat.com/governance (aufgerufen am 23. 12. 2024).

[76] L2Beats. *Loopring Risiko Analyse*. URL: https://l2beat.com/scaling/projects/loopring#risk-analysis (aufgerufen am 12. 12. 2024).

[77] L2Beats. *Taiko Risiko Analyse*. URL: https://l2beat.com/scaling/projects/taiko#risk-analysis (aufgerufen am 12. 12. 2024).

[78] LoopringTeam. (2023) *Mint NFT | Loopring Dev Docs*. URL: https://docs-protocol.loopring.io/counterfactual-nft/api-references/mint-nft (aufgerufen am 09. 12. 2024).

[79] OpenZeppelin. *ERC1155 - OpenZeppelin Docs*. URL: https://docs.openzeppelin.com/contracts/3.x/erc1155 (aufgerufen am 09. 12. 2024).

[80] Anonymous. *Directories - File systems | IPFS Docs*. URL: https://docs.ipfs.tech/concepts/file-systems/#create-a-directory (aufgerufen am 09. 12. 2024).

[81] LoopringTeam. (2023) *Introduction Counterfactual NFTs | Loopring Dev Docs*. URL: https://docs-protocol.loopring.io/counterfactual-nft/introduction (aufgerufen am 09. 12. 2024).

[82] OpenZeppelin. *ERC 1155 - setApprovalForAll*. URL: https://docs.openzeppelin.com/contracts/3.x/api/token/erc1155#IERC1155-setApprovalForAll-address-bool- (aufgerufen am 09. 12. 2024).

[83] OpenZeppelin. *ERC 1155 - saveBatchTransferFrom*. URL: https://docs.openzeppelin.com/contracts/3.x/api/token/erc1155#IERC1155-safeBatchTransferFrom-address-address-uint256---uint256---bytes- (aufgerufen am 09. 12. 2024).

[84] OpenZeppelin. *ERC 1155 - saveTransferFrom*. URL: https://docs.openzeppelin.com/contracts/3.x/api/token/erc1155#IERC1155-safeTransferFrom-address-address-uint256-uint256-bytes- (aufgerufen am 09. 12. 2024).

[85] OpenZeppelin. *EnumerableSet- OpenZeppelin Docs*. URL: https://docs.openzeppelin.com/contracts/3.x/api/utils#EnumerableSet (aufgerufen am 09. 12. 2024).

[86] OpenZeppelin. *ERC 1155 - TransferSingle*. URL: https://docs.openzeppelin.com/contracts/3.x/api/token/erc1155#IERC1155-TransferSingle-address-address-address-uint256-uint256- (aufgerufen am 09. 12. 2024).

[87] OpenZeppelin. *ERC 1155 - TransferBatch*. URL: https://docs.openzeppelin.com/contracts/3.x/api/token/erc1155#IERC1155-TransferBatch-address-address-address-uint256---uint256--- (aufgerufen am 09. 12. 2024).

[88] Anonymous. *Loopring's Layer-2 DEX Scaling Solution: OCDA*. URL: https://www.gemini.com/cryptopedia/loopring-coin-dex-decentralized-exchange#section-loopring-crypto-tech-unboxed-ocda-ring-miners-order-rings-order-sharing (aufgerufen am 09. 12. 2024).

[89] LoopringTeam. *Loopring Design Doc - Throughput*. URL: https://github.com/Loopring/protocols/blob/master/packages/loopring_v3/DESIGN.md#throughput (aufgerufen am 09. 12. 2024).

[90] TaikoLabs. *Inception layers*. URL: https://docs.taiko.xyz/core-concepts/inception-layers/ (aufgerufen am 09. 12. 2024).

[91] taikoscan.io. *Taiko Mainnet Daily Transactions Chart*. URL: https://taikoscan.io/chart/tx (aufgerufen am 09. 12. 2024).

[92] LoopringTeam. *Documentation of the Loopring SDK*. URL: https://loopring.github.io/loopring_sdk/ (aufgerufen am 27. 12. 2024).

[93] Anonymous. *eddsa-metamask-plugin/README.md at master · Loopring/eddsa-metamask-plugin*. URL: https://github.com/Loopring/eddsa-metamask-plugin/blob/master/README.md (aufgerufen am 09. 12. 2024).

[94] ethers.js Project. *ethers v6 Documentation*. URL: https://docs.ethers.org/v6/ (aufgerufen am 09. 12. 2024).

[95] LoopringTeam. *Loopring Design Document - Account Creation*. URL: https://github.com/Loopring/protocols/blob/master/packages/loopring_v3/DESIGN.md#account-creation (aufgerufen am 09. 12. 2024).

[96] LoopringTeam. *JubJub Curve*. URL: https://github.com/Loopring/loopring_sdk/blob/master/src/api/sign/poseidon/jubjub.ts (aufgerufen am 09. 12. 2024).

[97] LoopringTeam. *BabyJub Curve*. URL: https://github.com/Loopring/loopring_sdk/blob/master/src/api/sign/poseidon/babyJub.ts (aufgerufen am 09. 12. 2024).

[98] Barry WhiteHat et al. *ERC-2494: Baby Jubjub Elliptic Curve*. URL: https://eips.ethereum.org/EIPS/eip-2494 (aufgerufen am 09. 12. 2024).

[99] LoopringTeam. (2023) *EdDSA sign | Loopring Dev Docs*. URL: https://docs-protocol.loopring.io/resources/signature/eddsa-signature/eddsa-sign (aufgerufen am 09. 12. 2024).

[100] LoopringTeam. (2023) *Get apiKey | Loopring Dev Docs*. URL: https://docs-protocol.loopring.io/loopring-account/api-references/get-apikey (aufgerufen am 09. 12. 2024).

[101] LoopringTeam. (2023) *Create collection | Loopring Dev Docs*. URL: https://docs-protocol.loopring.io/counterfactual-nft/api-references/nft-collection/create-collection (aufgerufen am 09. 12. 2024).

[102] LoopringTeam. (2023) *Sample code | Loopring Dev Docs*. URL: https://docs-protocol.loopring.io/counterfactual-nft/api-references/nft-collection/create-collection/sample-code (aufgerufen am 09. 12. 2024).

[103] LoopringTeam. (2023) *Sample code | Loopring Dev Docs*. URL: https://docs-protocol.loopring.io/counterfactual-nft/api-references/mint-nft/sample-code (aufgerufen am 09. 12. 2024).

[104] LoopringTeam. *loopring_sdk/docs/js_sdk/NFTAction/metaNFT.md at master · Loopring/loopring_sdk*. URL: https://github.com/Loopring/loopring_sdk/blob/master/docs/js_sdk/NFTAction/metaNFT.md (aufgerufen am 09. 12. 2024).

[105] LoopringTeam. (2023) *Storage Id | Loopring Dev Docs*. URL: https://docs-protocol.loopring.io/resources/storage-id (aufgerufen am 09. 12. 2024).

[106] LoopringTeam. *Loopring Design Document - Storage*. URL: https://github.com/Loopring/protocols/blob/master/packages/loopring_v3/DESIGN.md#storage (aufgerufen am 09. 12. 2024).

[107] LoopringTeam. (2024) *Validate NFT Order | Loopring Dev Docs*. URL: https://docs-protocol.loopring.io/counterfactual-nft/api-references/validate-nft-order (aufgerufen am 09. 12. 2024).

[108] LoopringTeam. (2024) *Trade NFT | Loopring Dev Docs*. URL: https://docs-protocol.loopring.io/counterfactual-nft/api-references/trade-nft (aufgerufen am 09. 12. 2024).

[109] ethers.js Project. *Wallet - ethers docs*. URL: https://docs.ethers.org/v6/api/wallet/#Wallet (aufgerufen am 09. 12. 2024).

[110] ethers.js Project. *Contract - ethers docs*. URL: https://docs.ethers.org/v5/api/contract/contract/ (aufgerufen am 09. 12. 2024).

[111] ethers.js Project. *Contract populateTransaction - ethers docs*. URL: https://docs.ethers.org/v5/api/contract/contract/#contract-populateTransaction (aufgerufen am 09. 12. 2024).

[112] ethers.js Project. *Signers sendTransaction - ethers docs*. URL: https://docs.ethers.org/v6/api/providers/#Signer-sendTransaction (aufgerufen am 09. 12. 2024).

[113] ethers.js Project. *Contract transactionResponse .wait . ethers docs*. URL: https://docs.ethers.org/v6/api/contract/#ContractTransactionResponse-wait (aufgerufen am 09. 12. 2024).

[114] ethers.js Project. *Contract TransactionReceipt - ethers docs*. URL: https://docs.ethers.org/v6/api/contract/#ContractTransactionReceipt (aufgerufen am 09. 12. 2024).

[115] ethers.js Project. *interface parseLog - ethers docs*. URL: https://docs.ethers.org/v6/api/abi/#Interface-parseLog (aufgerufen am 09. 12. 2024).

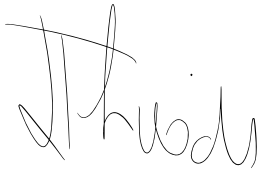
[116] ethers.js Project. *eventLogs - ethers docs*. URL: https://docs.ethers.org/v6/api/contract/#EventLog (aufgerufen am 09. 12. 2024).

[117] OpenZeppelin. *IERC1155-setApprovalForAll - OpenZeppelin Docs*. URL: https://docs.openzeppelin.com/contracts/4.x/api/token/erc1155#IERC1155-setApprovalForAll-address-bool- (aufgerufen am 09. 12. 2024).

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.



Mittweida, 31. Dezember 2024