

Proposal: crosschain messaging protocol

Development and launch plan for an interoperability feature in a crypto platform

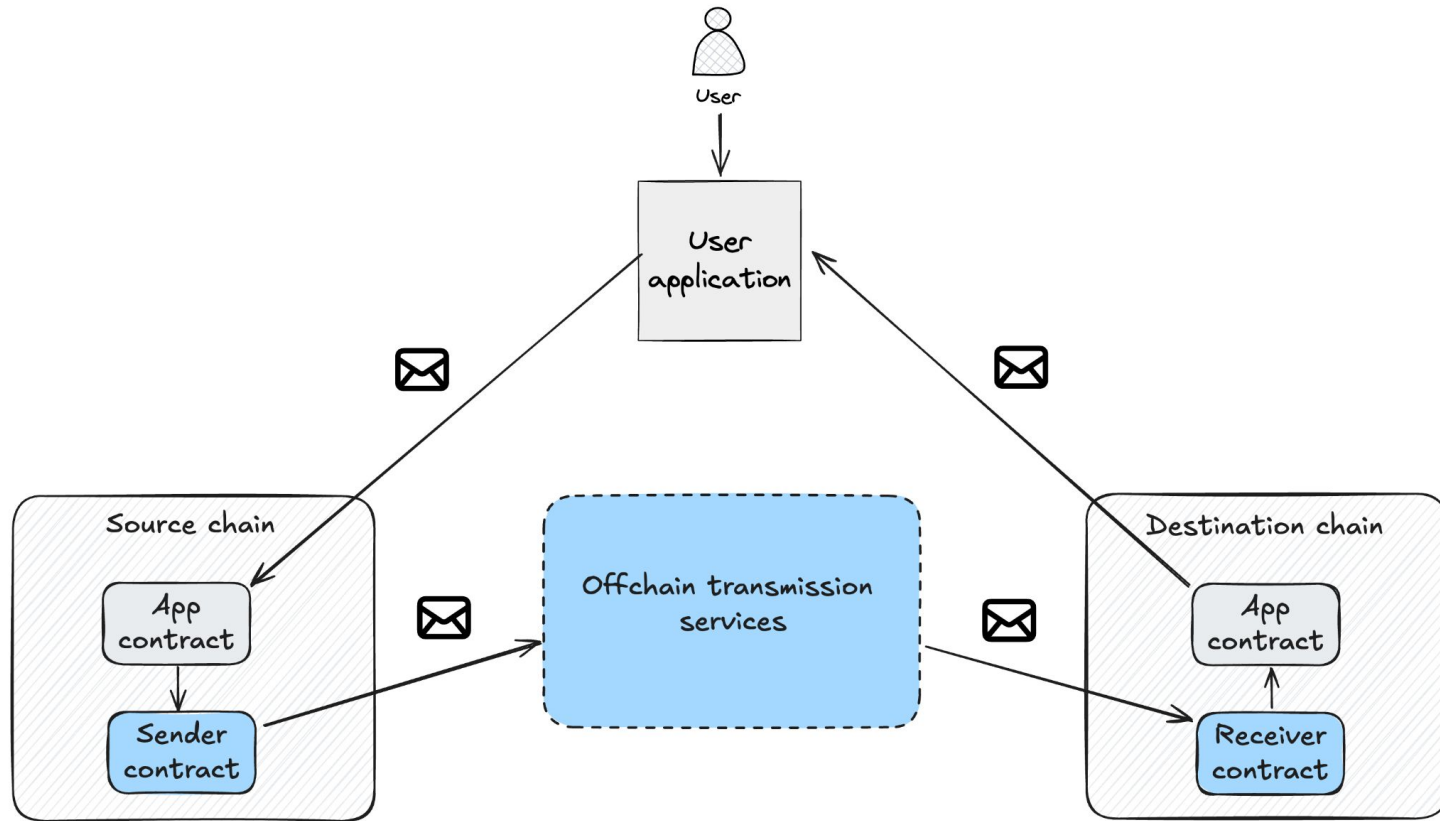
August 2024
André Zommerfelds
azommerfelds@gmail.com

Content

1. **Feature specification**
2. **Development plan**
3. **Sprint planning**
4. **Integration and testing**
5. **Documentation and tutorials**
6. **Launch strategy**
7. **Customer and developer feedback**

Introduction

General implementation of crosschain messaging protocols



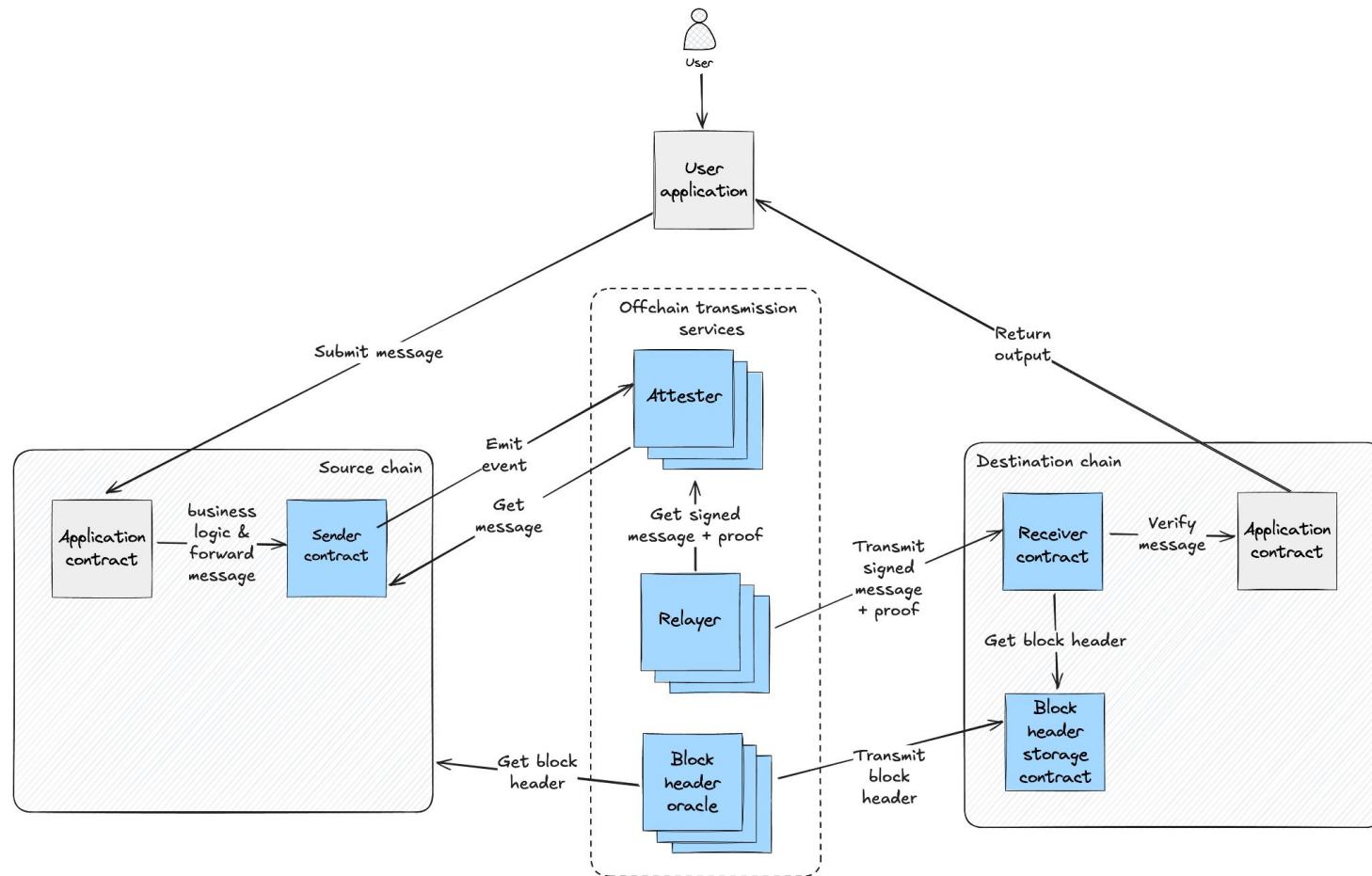
Source: own illustration

Message flow:

1. **User:** submits message with instructions to source chain smart contract
2. **Source chain & sender contract:** accepts & stores message in a block
3. **Transmission services:** sends package with necessary proofs to destination chain
4. **Destination chain & receiver contract:** receives package, verifies proofs, and forwards instructions to any other smart contract containing further business logic

1. Feature specification

Proposed architecture (high-level)



Message flow:

1. **User** submits a message
2. **Application contract (source chain)** executes any business logic and passes the message to the transmission contract
3. **Sender contract** emits a corresponding event
4. **Attester** listens to event and packages message with a proof
5. **Relayer** takes & forwards signed attestation
6. **Receiver contract** validates received package by verifying merkle proof and comparing block headers using the **block header storage contract** (data from **oracle**)
7. **Application contract (destination chain)** takes validated message and executes logic

1. Feature specification

Key functionalities & user stories

Component	Function	User story
Sender contract	Integrates with smart contracts to emit and store messages on source chain	As a developer, I want to integrate my smart contract with a contract to securely handle crosschain messages
Attester	Generates and signs merkle proof to verify message validity	As a developer, I want transmitted message to be trusted and verifiable
Relayer	Delivers message package and facilitates modular crosschain messaging	As a developer, I want protocol to be modular, with different nodes focusing on transmission and attestation
Receiver contract	Verifies merkle proofs and block headers on destination chain	As a developer, I want message on destination chain to be verified for authenticity
Block header oracle	Provides secure and timely block headers from source chain	As a developer, I want secure and accurate block headers for message validation
Block header storage contract	Stores and provides trusted block headers for validation	As a developer, I want secure and accurate block headers for message validation

Reasons for chosen architecture:

- Separation of attesters and relayers (modularity)
- Separate block header oracle (security & trust)
- Zero-knowledge proofs from block header oracles (speed & costs)

2. Development plan

Project size and timeline

Phase	Subtasks	Person weeks (est.)
Product design	Technical architecture	2
	Economics / Incentives design	2
	Requirements specification	2
Development	Sender contract	3
	Receiver contract	3
	Attester node	3
	Relayer node	2
	Block header oracle	4
	Block header storage contract	4
Audit	Security audit	3
Testing	Testing (unit, integration, security)	4
Rollout	Rollout buffer (local-testnet-mainnet, monitoring, bug fix, etc.)	4
Total person weeks		36
Total est. duration with 4 devs, 1 tester, 1 PM (person weeks)		22

3. Sprint planning

Sprint 1 (2 weeks)				
Component	User story	Requirement	Identifier	PBI
Sender contract	As a developer, I want to be able to connect smart contracts from different chains by integrating with a crosschain messaging protocol.	Functional	sender-contract-1	Contract interface: accept messages from application contract after executing necessary business logic.
Sender contract		Functional	sender-contract-2	Emit an event with message details, transaction hash, and metadata for attester.
Sender contract		Functional	sender-contract-3	Store message and associated data until successfully processed by attester and relayer.
Sender contract		Non-functional	sender-contract-4	Implement security features such as message nonce (replay attack)
Sender contract		Non-functional	sender-contract-5	Optimize event emission and storage for gas efficiency.

Sprint 2 (2 weeks)				
Component	User story	Requirement	Identifier	PBI
Receiver contract

Criteria for moving tasks from the backlog to the sprint:

- Criticality of features
- Readiness of design and specifications
- Path dependencies
- Risks and challenges
- Stakeholder requirements

4. Integration and testing

Testing strategy

Environments:

1. Local blockchains (e.g. using Hardhat Network)
2. Testnet (e.g. Sepolia on Ethereum/Arbitrum)
3. Mainnet

Testing dimensions:

- Functionality testing
- Security testing (incl. fuzz testing)
- Onchain costs
- Performance testing
- Load & stress testing
- Regression testing (in case of changes)

Testing plan

1. **Unit testing:** validate individual functions and components
2. **Security testing:** detect vulnerabilities, stress testing, and code audits
3. **Integration testing:** ensure different components interact correctly across components
4. **E2E testing:** validate the entire business workflow from start to finish
5. **User acceptance testing (UAT):** check with a real business application whether the protocol meets business needs and user/developer expectations, especially regarding performance, cost, and security.

5. Documentation and tutorials

Documentation structure

Section	Content
Introduction	Overview
Getting started	Quick start guide
	Supported networks
Core concepts	Architecture overview
	Security model
Guides	Basic integration
	Advanced features
Deployment	Deploying to testnet and mainnet
	Contract upgrades
API reference	Smart contract AB
	Relayer and attester interfaces
Resources	Tutorials and examples
	Glossary
	Support

Tutorial structure example

Section	Content
Introduction	Purpose and prerequisites
	Tools installation
	Clone repository
Setup	Deploy contracts
	Burn contract
	Mint contract
Implementing token contracts	Event listeners
	Sender contract integration
	Attester
Crosschain messaging	Relayer
	Test/mainnet deployment
Deploying to test or mainnet	Test/mainnet execution

6. Launch strategy

Phase	Marketing plans	Key milestones	Steps for success
Testnet launch; First 3-4 months	<ul style="list-style-type: none"> - Pre-launch announcement on crypto social (X, Farcaster, TG, Reddit, Discord) and developer platforms (e.g. GitHub) - Publish technical blog posts on Medium, Mirror, Paragraph - Engage with crypto communities on crypto social, do AMAs - Organize webinars/podcasts in crypto social 	<ul style="list-style-type: none"> - Core development completed - Security audits passed - Testnet launched - Onboarded early devs/builders 	<ul style="list-style-type: none"> - Finalize core features and test on testnets - Deploy contracts and invite early developers - Initially, attester and relayer will be centralized and permissioned for security reasons - Monitor performance, gather feedback, and resolve issues
Mainnet launch; Next 3-4 months	<p>Same as above, additionally:</p> <ul style="list-style-type: none"> - Launch partnerships with established crypto projects - Participate in major crypto events (e.g., ETHGlobal, Consensus, Devcon) - Host developer workshops and hackathons to promote protocol adoption 	<ul style="list-style-type: none"> - Implemented feedback from testnet launch - Mainnet launched - First apps built using protocol 	<ul style="list-style-type: none"> - Address testnet feedback and finalize product - Scale infrastructure, ensure robustness, and prepare for decentralization - Launch marketing campaign and open access to all developers
Decentralization; Next 3-4 months	Decentralization roadmap: announcements on social media, blog posts, and community engagement	<ul style="list-style-type: none"> - Attesters and relayers are permissionless - Fully open-source code 	<ul style="list-style-type: none"> - Open-source all code - Progressively opening attester and relayer network - If above OK, make them permissionless

7. Customer and developer feedback

Strategy to gather and incorporate feedback

- **Define target customers:**
 - Blockchain developers
 - Sophisticated crypto builders
- **Define & prioritize opportunities:**
 - Understand main needs, problems, or desires
 - Save them with user story / experience mapping
- **Continuous client interaction:**
 - Weekly interviews
 - Build in the open: engage with the community
- **Incorporate feedback:**
 - See “Analyzing and integrating feedback”
- **Channels to be used:**
 - Discord/TG for technical Q&A; GitHub for open discussions; X for community-driven insights; Webinars/AMAs/podcasts for attention; hackathons to encourage innovation and active feedback; (bug) bounties for security feedback

Analyzing and integrating feedback

1. **Prioritization:**
 - a. Prioritize opportunities identified (from interviews & UX mapping):
 - i. Size: how many target customers requested/impacted
 - ii. Satisfaction: how important it is to their satisfaction
 - iii. Business: how well it aligns with our business strategy
 - iv. Market: how we want to position ourselves in the market
2. **Continuous discovery, continuous delivery:**
 - a. Feedback reviews
 - b. Rapid iterations
 - c. Adjust roadmap
 - d. Communicate changes & follow-up

Q&A

Appendix

Current crosschain messaging ecosystem

Categorizing crosschain messaging protocols

Protocol type	Details
State validating protocols (trustless)	<ul style="list-style-type: none">- Validation of state with full nodes- Hashed timelock contract (HTLC) for p2p asset exchange
Consensus verifying protocols (spectrum: trust-minimized -> trusted)	<ul style="list-style-type: none">- Validation of consensus with light-client in smart contract- Validation of consensus with light-client in chain of chains (e.g., IBC)- Validation with a ZKP + light clients using succinct properties (e.g., SNARKs)
Third-party attestation protocols (spectrum: trusted -> trust-minimized)	<ul style="list-style-type: none">- Multisig or threshold signature attestation- TEE, SGX attestation- Oracles / oracle networks attestation- Proof of authority (PoA) or proof of stake (PoS) attestation- Central authority attestation
Optimistic protocols (spectrum: trusted -> trust-minimized)	<ul style="list-style-type: none">- (Permissionless) Attestors + watchers for (permissionless) fault-proof (1 of N trust assumption)
Hybrid protocols	<ul style="list-style-type: none">- Combining character

Popular crosschain protocols:

- [LayerZero](#) (third-party attestation protocol)
- [Wormhole](#) (third-party attestation protocol)
- [Chainlink CCIP](#) (third-party attestation protocol)
- [Cosmos IBC](#) (consensus verifying protocol; onchain)
- [zkBridge / Polyhedra Network](#) (consensus verifying protocols; validity proof)
- [Hashi](#) (hybrid protocol; bridge aggregator)
- [Across](#) (hybrid protocol; settlement layer for asset transfer intents)

Source: overview inspired by the [Crosschain Risk Framework](#)