**Unlocking Blockchain Interoperability: Bridging the Gap Between Networks**

**Introduction**

Blockchain technology has introduced a paradigm shift in various industries, enabling decentralized applications, secure data sharing, and transparent transactions. However, the fragmentation of different blockchain networks has limited their potential to collaborate and communicate effectively. This is where the concept of blockchain interoperability comes into play. In this article, we will dive deep into the world of blockchain interoperability, exploring its significance, challenges, and the promising solutions that are reshaping the landscape.

**What is Blockchain Interoperability?**

Blockchain interoperability refers to the ability of different blockchain networks to communicate, share data, and collaborate seamlessly. This capability is crucial for realizing the full potential of blockchain applications, enabling cross-chain asset transfers, data sharing, and more. The drive towards interoperability presents both challenges and opportunities, paving the way for a more interconnected blockchain ecosystem.

**Why is it Important?**

**Efficient Cross-Chain Transactions**: Interoperability allows users to transfer assets between different blockchains without relying on centralized exchanges. Users can transact assets across different blockchain networks without needing to convert them to a base currency. This fosters liquidity, reduces transaction costs, and enhances the overall user experience.

**Expanding Decentralized Applications (DApps):** DApps often operate on specific blockchains, limiting their reach and potential. Interoperability enables DApps to leverage multiple blockchains’ unique capabilities, creating a more comprehensive and versatile ecosystem (e.g., Ethereum's smart contracts and Bitcoin's security).

**Scalability and Load Distribution**: By distributing transactions across interconnected blockchains, interoperability addresses the scalability issues that many networks face, enhancing overall performance and user satisfaction.

**Global Collaboration and Data Sharing**: Blockchain interoperability empowers organizations and industries to securely share data, collaborate on projects, and create new value propositions while maintaining data integrity and transparency.

A diagram of blockchain

Description automatically generated

**Approaches to Blockchain Interoperability**

There are several types of interoperability solutions that have been developed to enable cross-chain transactions and data sharing between different blockchain networks.

**Cross-Chain Bridges**

**A network of red circles and dots with Ice hockey rink in the background

Description automatically generated**

Cross-chain bridges act as intermediaries between blockchains, facilitating the secure transfer of assets and data. These bridges typically lock assets on the originating chain and mint an equivalent representation on the destination chain. When the user wants to return the asset, the bridge burns the representation on the destination chain and unlocks the original asset on the originating chain.

The most well-known interoperability solution to enable blockchain networks to interact is using a separate blockchain as a bridge to facilitate the exchange of data, assets and messages between different blockchain networks without the need for intermediaries, better known as cross-chain bridges

E.g.

* **Atomic Swaps:** Allow users to swap tokens from different blockchains without a centralized exchange. This is achieved through smart contracts that lock the tokens until both parties fulfill the swap conditions.
* **Wormhole**: Wormhole is a generic cross-chain bridge protocol that facilitates asset transfers between various blockchains like Ethereum, Solana, and Avalanche.

**Side chains**

Sidechains are separate and independent blockchains that are connected to the main blockchain through a two-way peg or bridge to improve the scalability by helping process some of the data from the main blockchain. This two-way peg acts as an intermediary which locks an asset in one blockchain to reserve it until the transfer to the other one is completed. Although sidechains have a connection to the parent blockchain, they use separate consensus algorithms and have different native tokens.

* **Liquid Network by Blockstream:** Liquid is a Bitcoin sidechain that allows for faster, confidential transactions and the issuance of digital assets. It uses a two-way peg to enable Bitcoin to move between the Bitcoin mainnet and the Liquid sidechain.

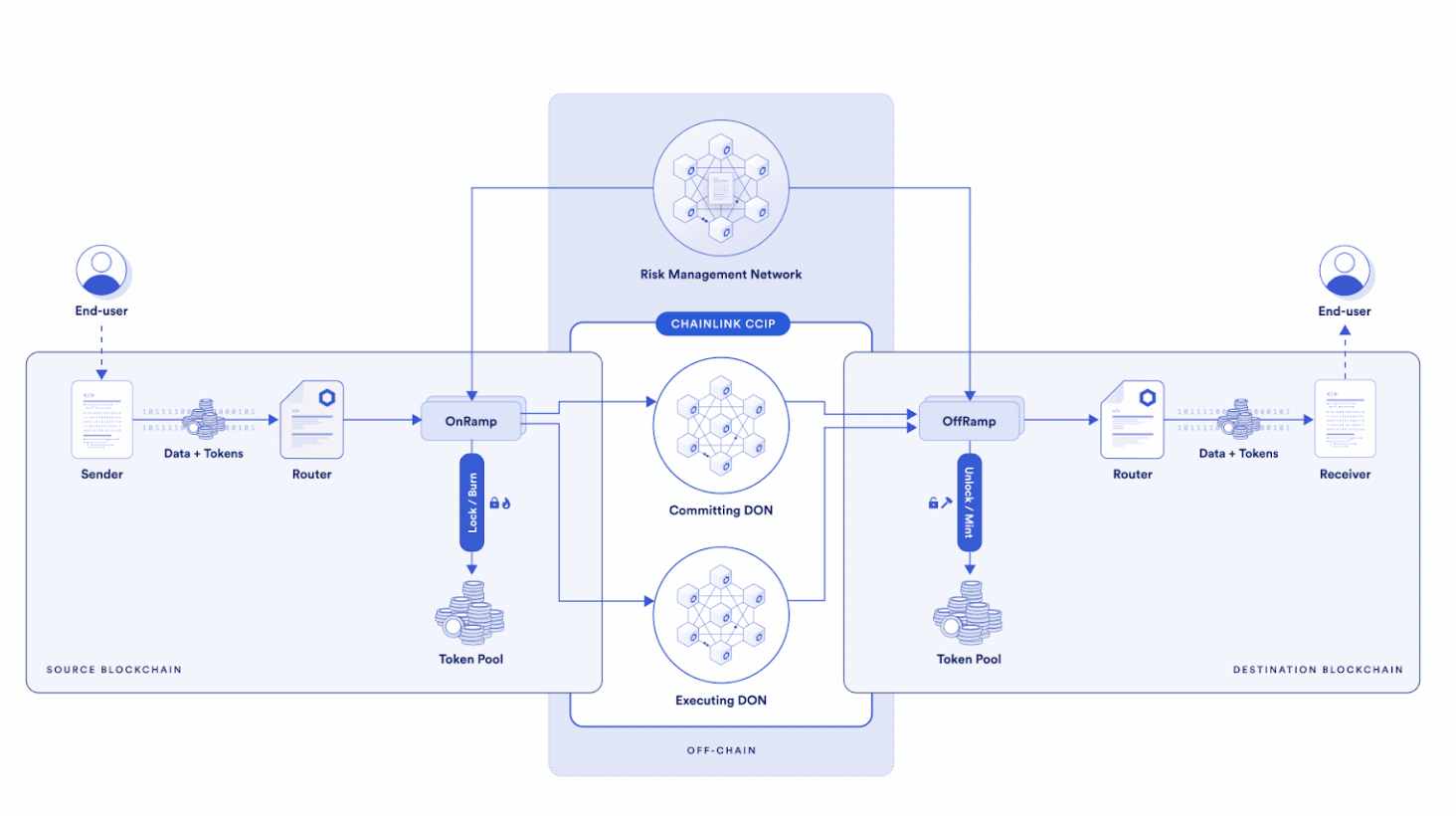
**Oracles**

An oracle is an agent or computer program that supports the transfer of data from external sources to the blockchain platform for on-chain usage. This is accomplished using smart contracts that add data/information about real-world scenarios to the blockchain platform. Once such data is transmitted within the blockchain platform, this data can be utilised to automate processes based on real business use cases. They are crucial for smart contracts that require real-world data to execute functions (e.g., fetching stock prices or weather data).

**Interoperability/Middleware Protocols**

Interoperability Protocols are software layers or frameworks that sit between different blockchain networks and facilitate communication and data sharing. These protocols abstract away the technical differences between blockchains and provide a standardised interface for interoperability. Interoperability protocols often include features such as data formatting, encoding, and consensus mechanisms that enable seamless interaction between different blockchains. They can support various types of interoperability, including data interoperability, asset interoperability, and smart contract interoperability.

[**Chainlink's CCIP**](https://chain.link/cross-chain)**(Cross-Chain Interoperability Protocol)** enables secure messaging and token transfers across multiple blockchain networks while using Chainlink's decentralized oracle network (DON) for trust minimization

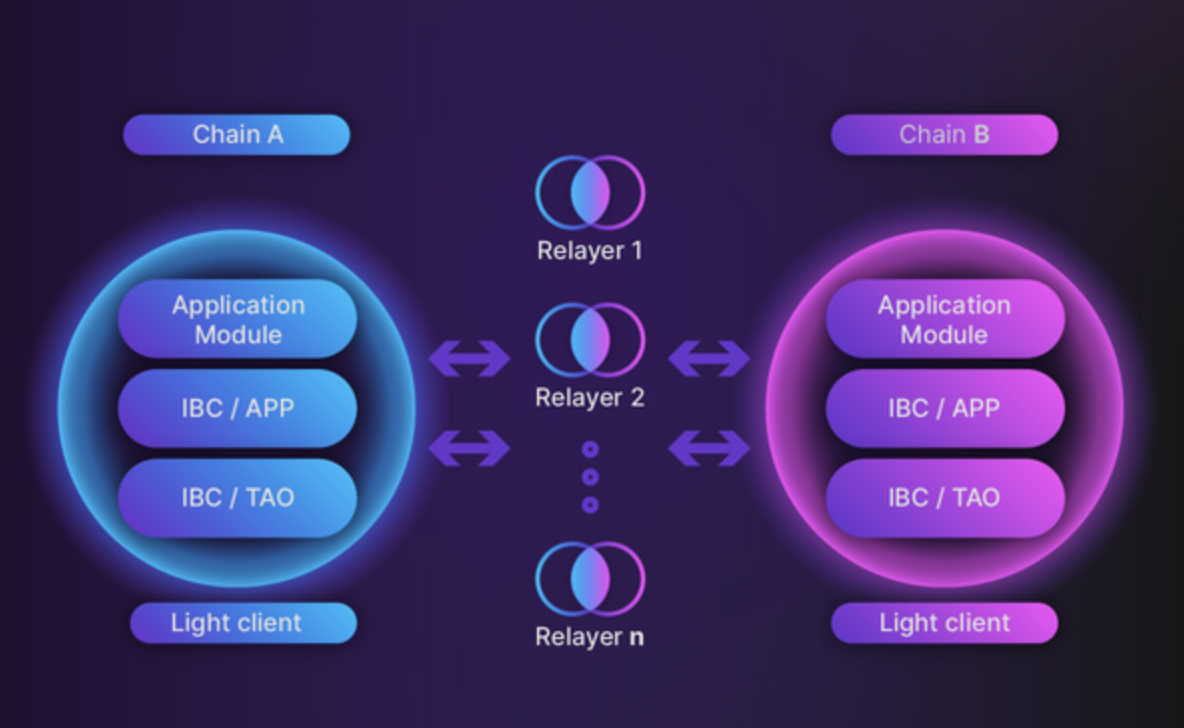


[**Wormhole**](https://wormhole.com/)enables seamless communication and asset transfers across over 30 blockchain networks, including Ethereum and Solana. For example, Lido, a leading liquid staking provider, used the Wormhole NTT Token infrastructure that leverages both Wormhole and Axelar Transceivers to build a bridge to ship tokens to BNB chain.

A screenshot of a computer

Description automatically generated

[**Cosmos' IBC**](https://tutorials.cosmos.network/academy/3-ibc/1-what-is-ibc.html)**(Inter-Blockchain Communication)** standardizes how independent blockchains communicate. It supports token transfers and general data exchange and enables chains to maintain independence while connecting to the broader Cosmos ecosystem.



**Axelar’s**[**General Message Passing (GMP)**](https://www.axelar.network/blog/general-message-passing-and-how-can-it-change-web3)protocol allows blockchain developers to call any function on any connected chain, enhancing interoperability and composability across the entire blockchain ecosystem. The cross-chain liquidity router Squid uses GMP to execute optimal cross-chain swaps.

A diagram of a network

Description automatically generated

**Exploring Chainlink's Cross-Chain Interoperability Protocol (CCIP)**

Chainlink CCIP is a protocol designed to facilitate secure and reliable communication between various blockchain networks. By leveraging Chainlink's decentralized oracle network, CCIP ensures that cross-chain transactions are executed with high levels of security and reliability. This protocol is set to revolutionize the way blockchains interact, opening up a myriad of possibilities for developers and users alike

[**Chainlink CCIP core capabilities**](https://docs.chain.link/ccip#chainlink-ccip-core-capabilities)

Chainlink CCIP supports three main capabilities:

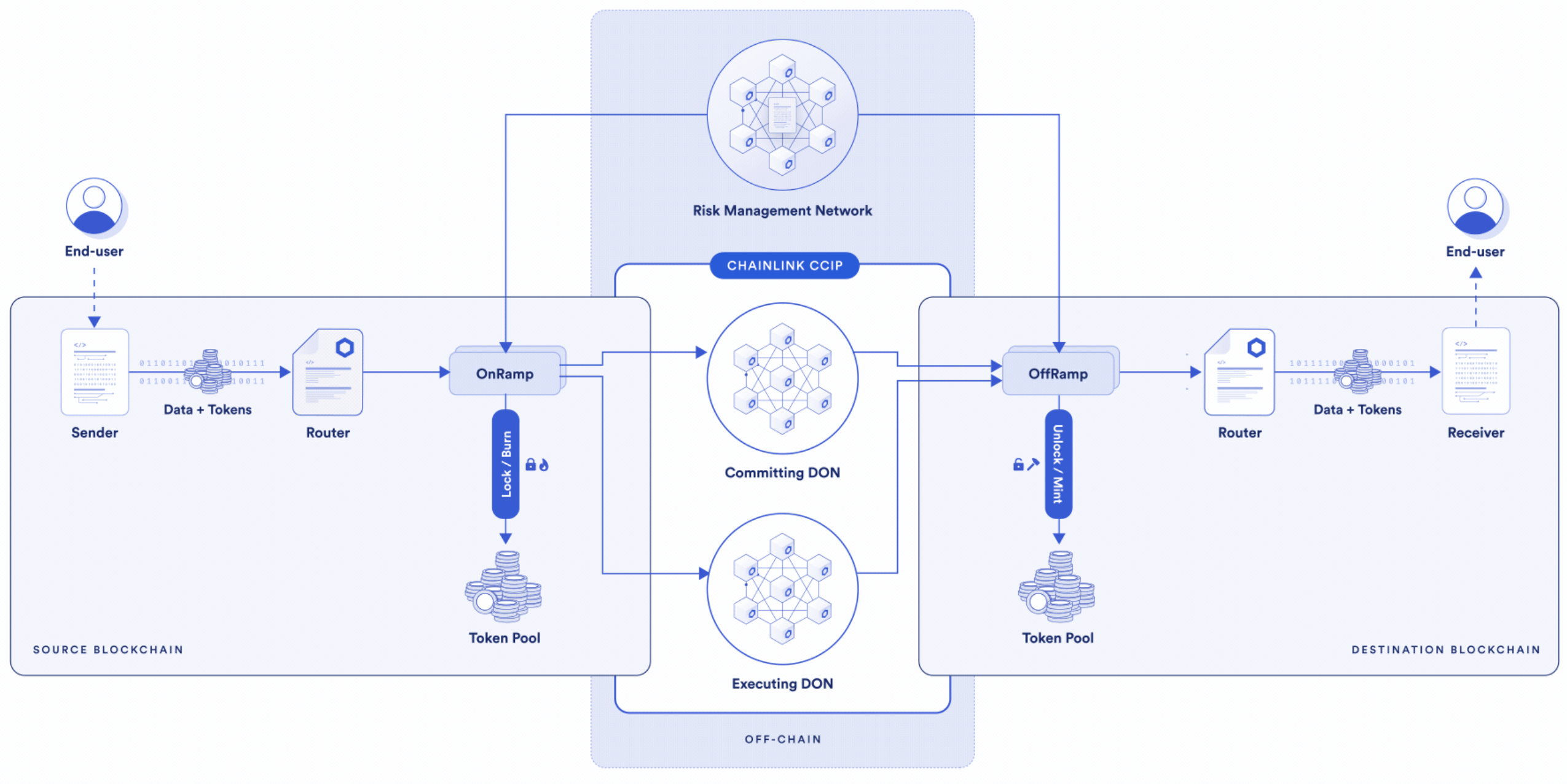
* Arbitrary Messaging: is the ability to send arbitrary data (encoded as bytes) to a receiving smart contract on a different blockchain. The developer is free to encode any data they wish to send. Typically, developers use arbitrary messaging to trigger an informed action on the receiving smart contract, such as rebalancing an index, minting a specific NFT, or calling an arbitrary function with the sent data as custom parameters. Developers can encode multiple instructions in a single message, enabling them to orchestrate complex, multi-step, multi-chain tasks.
* Token Transfer: You can transfer tokens to a smart contract or directly to an [Externally Owned Account(EOA)](https://ethereum.org/en/developers/docs/accounts/#types-of-account) on a different blockchain.
* Programmable Token Transfer: is the ability to simultaneously transfer tokens and arbitrary data (encoded as bytes) within a single transaction. This mechanism allows users to transfer tokens and send instructions on what to do with those tokens. For example, a user could transfer tokens to a lending protocol with instructions to leverage those tokens as collateral for a loan, borrowing another asset to be sent back to the user.

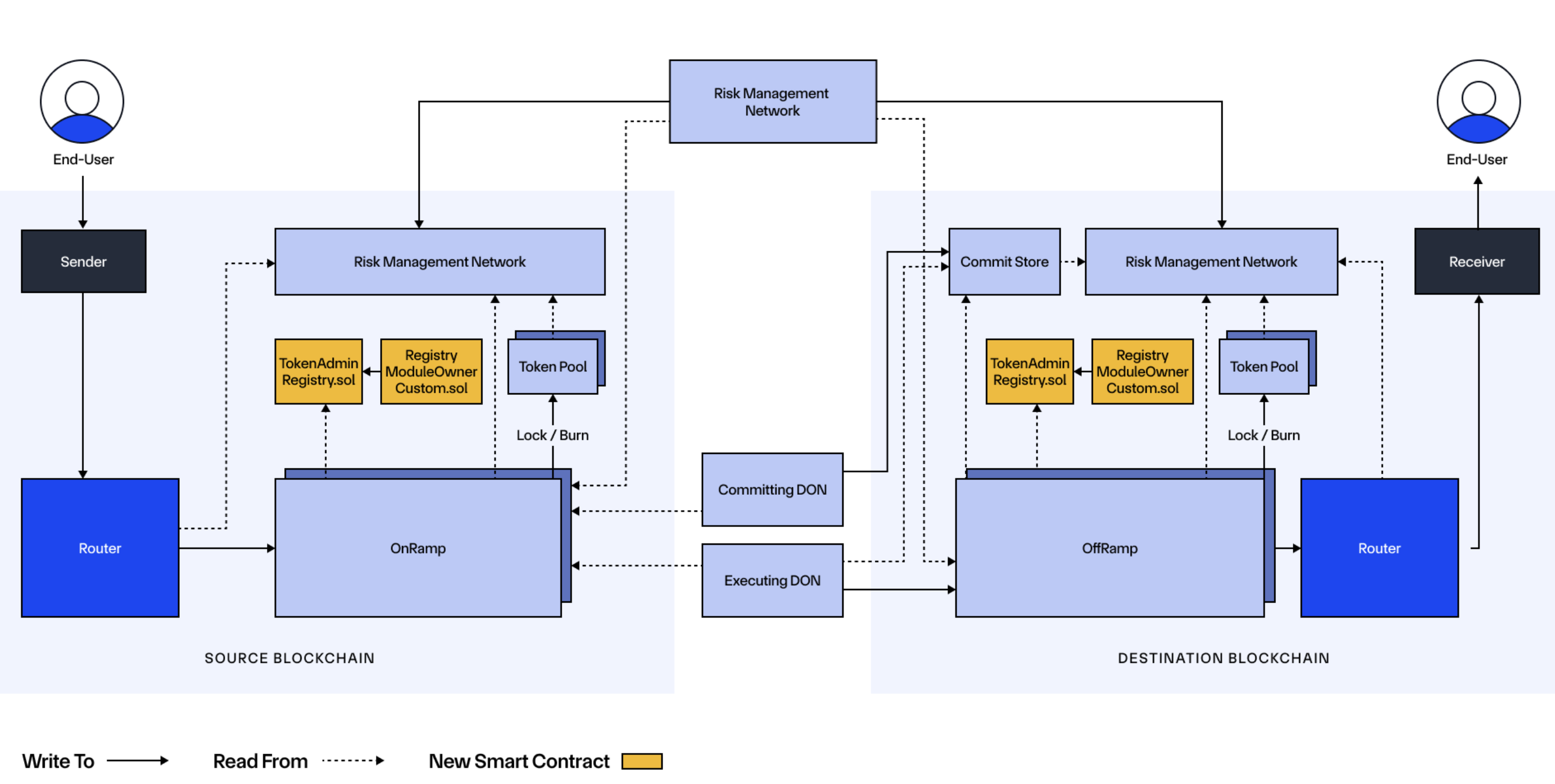
**Potential Use Cases**

The implementation of CCIP could lead to numerous innovative use cases, such as:

* Cross-Chain DeFi Applications: Decentralized finance (DeFi) applications can benefit significantly from CCIP by accessing liquidity and services across multiple blockchains. This can lead to more efficient markets and novel financial instruments.
* Multi-Chain NFTs: Non-fungible tokens (NFTs) could leverage CCIP to exist across multiple chains, allowing artists and creators to reach broader audiences and utilize different blockchain features.
* Cross-Chain Governance: DAOs and other governance systems can implement cross-chain voting and decision-making processes, enhancing inclusivity and participation from a wider range of stakeholders.

**CCIP Architecture**





**Onchain Components**

[Router](https://docs.chain.link/ccip/architecture#router)

The Router is the primary contract CCIP users interface with. This contract is responsible for initiating cross-chain interactions. One router contract exists per blockchain. When transferring tokens, callers have to approve tokens for the router contract. The router contract routes the instruction to the destination-specific [OnRamp](https://docs.chain.link/ccip/architecture#onramp).

When a message is received on the destination chain, the router is the contract that “delivers” tokens to the user's account or the message to the receiver's smart contract.

[Commit Store](https://docs.chain.link/ccip/architecture#commit-store)

The [Committing DON](https://docs.chain.link/ccip/architecture#committing-don) interacts with the CommitStore contract on the destination blockchain to store the Merkle root of the finalized messages on the source blockchain. This Merkle root must be blessed by the [Risk Management Network](https://docs.chain.link/ccip/concepts#risk-management-network) before the [Executing DON](https://docs.chain.link/ccip/architecture#executing-don) can execute them on the destination blockchain. The CommitStore ensures the message is blessed by the [Risk Management Network](https://docs.chain.link/ccip/architecture#risk-management-network) and only one CommitStore exists per [lane](https://docs.chain.link/ccip/concepts#lane).

[OnRamp](https://docs.chain.link/ccip/architecture#onramp)

One OnRamp contract exists per [lane](https://docs.chain.link/ccip/concepts#lane). This contract performs the following tasks:

* Checks destination blockchain-specific validity, such as validating account address syntax.
* Verifies message size limits and gas limits.
* Keeps track of sequence numbers to preserve the order of messages for the receiver
* Manages [billing](https://docs.chain.link/ccip/billing).
* Interacts with the [TokenAdminRegistry](https://docs.chain.link/ccip/architecture#tokenadminregistry-and-registrymoduleownercustom) to retrieve the token pool linked to a token if the message includes a token transfer.
* Interacts with the [TokenPool](https://docs.chain.link/ccip/architecture#token-pools) to lock or burn a specified amount of tokens if the message includes a token transfer.
* Emits an event monitored by the [committing DON](https://docs.chain.link/ccip/architecture#committing-don).

[OffRamp](https://docs.chain.link/ccip/architecture#offramp)

One OffRamp contract exists per [lane](https://docs.chain.link/ccip/concepts#lane). This contract performs the following tasks:

* Ensures the message is authentic by verifying the proof provided by the Executing DON against a committed and blessed Merkle root.
* Ensures transactions are executed only once.
* Interacts with the [TokenAdminRegistry](https://docs.chain.link/ccip/architecture#tokenadminregistry-and-registrymoduleownercustom) to retrieve the token pool linked to a token if the message includes a token transfer.
* If the CCIP transaction includes token transfers, the OffRamp contract calls the [TokenPool](https://docs.chain.link/ccip/architecture#token-pools) to unlock or mint the correct assets for the receiver.
* After validation, the OffRamp contract transmits any received message to the [Router](https://docs.chain.link/ccip/architecture#router) contract.

[TokenAdminRegistry and RegistryModuleOwnerCustom](https://docs.chain.link/ccip/architecture#tokenadminregistry-and-registrymoduleownercustom)

The registry contracts manage the registration of tokens within the Cross-Chains Tokens (CCT) standard. For more information, read the [CCT standard concepts](https://docs.chain.link/ccip/concepts/cross-chain-tokens).

[**Token pools**](https://docs.chain.link/ccip/architecture#token-pools)

Each token is associated with its own token pool, an abstraction layer over ERC-20 tokens designed to facilitate token-related operations for OnRamping and OffRamping. Token pools provide rate limiting, a security feature enabling token developers to set a maximum rate at which their token can be transferred per lane. Token pools are configured to lock or burn tokens on the source blockchain and unlock or mint tokens on the destination blockchain. This setup results in four primary mechanisms:

* Burn and Mint: Tokens are burned on the source blockchain, and an equivalent amount of tokens are minted on the destination blockchain.
* Lock and Mint: Tokens are locked on their issuing blockchain, and fully collateralized "wrapped" tokens are minted on the destination blockchain. These wrapped tokens can be transferred across non-issuing blockchains using the *Burn and Mint* mechanism.
* Burn and Unlock: Tokens are burned on the source blockchain, and an equivalent amount of tokens are released on the destination blockchain. This mechanism is the inverse of the *Lock and Mint* mechanism. It applies when you send tokens to their issuing source blockchain.
* Lock and Unlock: Tokens are locked on the source blockchain, and an equivalent amount of tokens are released on the destination blockchain.

Currently, token pools are administered in two ways:

* Self-Administered Token Pools: These are managed directly by projects, allowing them to self-manage their tokens. Projects configure their token pools autonomously. This is currently only possible on blockchains where the [Cross-Chain Token standard](https://docs.chain.link/ccip/concepts/cross-chain-tokens) is available.
* CCIP-Managed Token Pools: Some token pools that existed before the [Cross-Chain Token standard](https://docs.chain.link/ccip/concepts/cross-chain-tokens) are managed by CCIP and will continue to exist alongside the self-administered ones. Eventually, all token pools will be fully administered by their respective projects.

The mechanism for handling tokens varies depending on the characteristics of each token. Below are several examples to illustrate this:

* LINK Token is minted on a single blockchain (Ethereum mainnet) and has a fixed total supply. Consequently, CCIP cannot natively mint it on another blockchain. For LINK, the token pool is configured to lock tokens on Ethereum mainnet (the issuing blockchain) and mint them on the destination blockchain. Conversely, when transferring from a non-issuing blockchain to Ethereum mainnet, the LINK token pool is set to burn the tokens on the source (non-issuing) blockchain and unlock them on Ethereum Mainnet (issuing). For example, transferring 10 LINK from Ethereum mainnet to Base mainnet involves the LINK token pool locking 10 LINK on Ethereum mainnet and minting 10 LINK on Base mainnet. Conversely, transferring 10 LINK from Base mainnet to Ethereum mainnet involves the LINK token pool burning 10 LINK on Base mainnet and unlocking 10 LINK on Ethereum mainnet.
* Wrapped native Assets (e.g., WETH) use a *Lock and Unlock* mechanism. For instance, when transferring 10 WETH from Ethereum mainnet to Optimism mainnet, the WETH token pool will lock 10 WETH on Ethereum mainnet and unlock 10 WETH on Optimism mainnet. Conversely, transferring from Optimism mainnet back to Ethereum mainnet involves the WETH token pool locking 10 WETH on Optimism mainnet and unlocking 10 WETH on the Ethereum mainnet.
* Stablecoins (e.g., USDC) can be minted natively on multiple blockchains. Their respective token pools employ a *Burn and Mint* mechanism, burning the token on the source blockchain and then minting it natively on the destination blockchain.
* Tokens with a Proof Of Reserve (PoR) with a PoR feed on a specific blockchain present a challenge for the *Burn and Mint* mechanism when applied across other blockchains due to conflicts with the PoR feed. For such tokens, the *Lock and Mint* approach is preferred.

Transferring Native Gas Tokens

To transfer ETH, from one blockchain to another, follow these steps:

1. Wrap ETH into WETH: Users must first interact with the WETH contract or use a DEX to convert their ETH into Wrapped Ether (WETH).
2. Send WETH via CCIP: Next, users call the CCIP Router to send WETH to the receiver account on the desired destination blockchain.
3. Unwrap WETH back into ETH: Finally, on the destination blockchain, users call the WETH contract or use a DEX to convert the WETH into ETH.

Note: You can use the [EthSenderContract.sol](https://github.com/smartcontractkit/ccip/tree/release/contracts-ccip-1.5.1/contracts/src/v0.8/ccip/applications/EtherSenderReceiver.sol) contract as a reference to transfer ETH across blockchains using CCIP.

[Risk Management Network contract](https://docs.chain.link/ccip/architecture#risk-management-network-contract)

The Risk Management contract maintains the list of Risk Management node addresses that are allowed to bless or curse. The contract also holds the quorum logic for blessing a committed Merkle Root and cursing CCIP on a destination blockchain. Read the [Risk Management Network Concepts](https://docs.chain.link/ccip/concepts#risk-management-network) section to learn more.

**OffChain Components**

[**Committing DON**](https://docs.chain.link/ccip/architecture#committing-don)

The Committing DON has several jobs where each job monitors cross-chain transactions between a given source blockchain and destination blockchain:

* Each job monitors events from a given [OnRamp contract](https://docs.chain.link/ccip/architecture#onramp) on the source blockchain.
* The job waits for [finality](https://docs.chain.link/ccip/concepts#finality), which depends on the source blockchain.
* The job bundles transactions and creates a Merkle root. This Merkle root is signed by a quorum of oracles nodes part of the Committing DON.
* Finally, the job writes the Merkle root to the [CommitStore contract](https://docs.chain.link/ccip/architecture#commit-store) on the given destination blockchain.

[**Executing DON**](https://docs.chain.link/ccip/architecture#executing-don)

Like the [Committing DON](https://docs.chain.link/ccip/architecture#committing-don), the Executing DON has several jobs where each executes cross-chain transactions between a source blockchain and a destination blockchain:

* Each job monitors events from a given [OnRamp contract](https://docs.chain.link/ccip/architecture#onramp) on the source blockchain.
* The job checks whether the transaction is part of the relayed Merkle root in the [CommitStore contract](https://docs.chain.link/ccip/architecture#commit-store).
* The job waits for the [Risk Management Network](https://docs.chain.link/ccip/architecture#risk-management-network) to bless the message.
* Finally, the job creates a valid Merkle proof, which is verified by the [OffRamp contract](https://docs.chain.link/ccip/architecture#offramp) against the Merkle root in the [CommitStore contract](https://docs.chain.link/ccip/architecture#commit-store). After these check pass, the job calls the [OffRamp contract](https://docs.chain.link/ccip/architecture#offramp) to complete the CCIP transactions on the destination blockchain.

Separating commitment and execution permits the [Risk Management Network](https://docs.chain.link/ccip/architecture#risk-management-network) to have enough time to check the commitment of messages before executing them. The delay between commitment and execution also permits additional checks such as abnormal reorg depth, potential simulation, and slashing.

Saving a commitment is compact and has a fixed gas cost, whereas executing user callbacks can be highly gas intensive. Separating commitment and execution permits execution by end users in various cases, such as retrying failed executions.

[**Risk Management Network**](https://docs.chain.link/ccip/architecture#risk-management-network)

The Risk Management Network is a set of independent nodes that monitor the Merkle roots committed by the [Committing DON](https://docs.chain.link/ccip/architecture#committing-don) into the [Commit Store](https://docs.chain.link/ccip/architecture#commit-store).

Each node compares the committed Merkle roots with the transactions received by the [OnRamp contract](https://docs.chain.link/ccip/architecture#onramp). After the verification succeeds, it calls the Risk Management contract to *"bless"*  the committed Merkle root. When there are enough blessing votes, the root becomes available for execution. In case of anomalies, each Risk Management node calls the Risk Management contract to *"curse"*  the system. If the cursed quorum is reached, the Risk Management contract is paused to prevent any CCIP transaction from being executed.