Ēnosys Bridge Whitepaper

March 27, 2023

Abstract

A distributed ledger provides a platform on which users can transact and build applications. The unified environment, that each ledger offers, allows users to seamlessly interact with all deployed applications. In addition, it permits information and digital assets to be transferred between applications. Nonetheless, the blockchain ecosystem comprises multiple distributed ledgers, each with their own community of users and services. This fragmentation often results in siloed environments, where users and applications can interact only within the confines of their ledger. In this paper, we describe how to allow cross-chain interactions and transfers of digital assets. In particular, we describe a mechanism for transferring assets across ledgers in the form of "wrapped" assets. We distill the functionality of this bridge between ledgers and highlight the roles of the key participants on each stage of the wrapping process.

1 Introduction

With the introduction of Bitcoin [Nak08], an ecosystem of financial services was born. The principal proposition of the new paradigm that Bitcoin offered was the decentralization of database infrastructure, on top of which financial applications could be built. The decade since Bitcoin's inception saw the creation of hundreds of systems that followed on its steps and expanded the original design.

Out of all Bitcoin successors, Ethereum [W$^+$14] has arguably been the most used. Ethereum's innovation lied in its Turing complete virtual machine (EVM), which offered the ability to develop applications without programming language restrictions (as opposed to Bitcoin's limited scripting language). As a result, Ethereum has evolved as the primary hub for "Decentralized Finance", i.e., financial applications that run on top of distributed ledgers.[1]

Despite its success in terms of adoption, Ethereum has demonstrated limitations in terms of performance. On multiple occasions, Ethereum's network has been congested to a point of near unusability. At these times, the delay until transactions are finalized reached many hours and the fees for interacting

[1] Such applications include exchanges, marketplaces, digital asset creators, etc. For more information we refer to [WPG$^+$21].

with the ledger reached tens or even hundreds of USD per transaction. Consequently, a number of systems were created, which offered EVM-compatible services and aimed at offering better scalability guarantees and acquiring some of Ethereum's usage traffic.

With the introduction of these systems though, a new problem came about. By default, each distributed ledger is oblivious to other ledgers. The maintainers of a ledger, which run the full nodes, are typically required to keep a copy only of the ledger (which they maintain). All information recorded on the ledger should be self-sustained and adhere to the ledger's validity rules. However, since the different ledgers form an ecosystem, users often interact with applications and maintain assets across various ledgers. Therefore, enabling the coordinated usage and exchange of information and assets from various ledgers is a powerful mechanism for the evolution of the ecosystem and a qualitative improvement of the offered services.

In this work, we describe a mechanism for transferring assets across EVM-compatible ledgers. In particular, we focus on ERC-20 assets,[2] i.e., fungible digital tokens. In the following sections, we describe the creation of "bridges", which enable assets to be moved back and forth across compatible ledgers. Specifically, Section 2 describes the smart contracts that form the two sides of a wrap bridge, whereas Section 3 highlights the different roles that key parties play in the system's execution.

2 A Bridge of Wrapped Assets

In this section, we cover the process of bridging assets across distributed ledgers. The principal element of the mechanism is creating "wrapped" assets on the receiving chain, which act as representations of the assets transferred from the originating chain. For ease of readability, we will denote by $\mathcal{L}_A$ the originating chain and by $\mathcal{L}_B$ the receiving chain. In principle, any EVM-compatible ledger could serve as originating or receiving.

The act of bridging begins on $\mathcal{L}_A$. On this ledger, users (via their addresses and the public keys that control them) manage digital assets. In this work, we only consider ERC-20 assets. ERC-20 is the most-used standard for creating fungible tokens on EVM-compatible ledgers. A token of this type is created and managed by a smart contract on a specific ledger.[3] The distribution of ownership of an ERC-20 token, meaning which users (accounts or addresses) control which amounts of the token, is defined in the smart contract that controls the token and is publicly available. Therefore, by default, each token exists only within the ledger on which its smart contract lives.

To transfer an amount of an ERC-20 token to a different ledger $\mathcal{L}_B$, there should exist a smart contract (on $\mathcal{L}_B$) which will control the transferred to-

[2] `https://eips.ethereum.org/EIPS/eip-20`

[3] Here, when we refer to an ERC-20 token, we refer to all (fungible) tokens of the same type. Each such (family of) tokens is distinguished by unique identifiers, namely a name and a symbol.

kens. This contract should offer the same functionalities defined in the ERC-20 standard (and possibly some extra, bridging-related operations, discussed below). The tokens controlled by this contract are "wrapped" assets. Intuitively, a wrapped asset is a representation (on $\mathcal{L}_B$) of an asset that was originally created on another ledger ($\mathcal{L}_A$).

This representation should be 1-1. Specifically, for a number of wrapped assets on $\mathcal{L}_B$, there should have been deposited to the bridge an equal amount of original assets on $\mathcal{L}_A$. Importantly, a wrapped asset and its (original) counterpart should not be used simultaneously. Therefore, at any point in time, if the wrapped asset is in circulation (s.t. its owner can transfer or use it in any way), the counterpart asset should be frozen. Equivalently, when a user wants to transfer an asset back to its original chain, the wrapped asset should be destroyed.

After a wrapped ERC-20 smart contract is created, a user should be able to transfer assets between ledgers. This process is conducted via a "bridge", which consists of two smart contracts on either side of the transfer (i.e., on the two ledgers, $\mathcal{L}_A$ and $\mathcal{L}_B$).

Briefly, a user deposits its tokens to the bridge contract on $\mathcal{L}_A$. For each deposit, a request for creating an equivalent number of wrapped assets on $\mathcal{L}_B$ is initiated. In its deposit, the user defines the address on $\mathcal{L}_B$, which will receive the wrapped assets. Following, the bridge contract on $\mathcal{L}_B$ generates a number of wrapped assets (equal to the number of deposited original assets) and assigns them to the user-defined address. The $\mathcal{L}_A$ bridge contract keeps the deposited assets in escrow, while the wrapped assets are freely used and transferred between $\mathcal{L}_B$ addresses. When a user wants to transfer assets back from $\mathcal{L}_B$ to $\mathcal{L}_A$, they deposit their wrapped tokens to the $\mathcal{L}_B$ bridge contract and, following a similar process as before, the wrapped assets are destroyed and the original assets on $\mathcal{L}_A$ are released.

Below, we describe the functionalities of both smart contracts and the details of transferring assets across the bridge.

2.1 Bridge Functionality

The smart contracts on either side of the bridge share, for the most part, the same functionality. For ease of reading and without loss of generality, we assume here that the sending side is ledger $\mathcal{L}_A$ and the receiving side is $\mathcal{L}_B$, therefore original assets (on $\mathcal{L}_A$) are transformed to wrapped assets (on $\mathcal{L}_B$); nonetheless, the same functionality holds for transforming wrapped assets back to their original.

First, the contract on $\mathcal{L}_A$ allows users to deposit assets to it. When a user deposits assets, the bridging process is initiated. We stress that, after a deposit is made, a user cannot revert it. Instead, they have to wait until the assets are transferred to the other side and then initiate a new transfer back, if they desire.

For each deposit, an event is emitted, containing its details. Specifically, the event defines: i) id: a unique id for the deposit; ii) N: the amount of tokens;

iii) α_{dest}: the destination address (which will receive assets on the other side of the bridge). Special parties, the *validators*, listen for these events.

After a validator V listens to an event for a deposit on $\mathcal{L}_A$, V initiates a request on the bridge contract on $\mathcal{L}_B$. The request contains the necessary information about the deposit, i.e., the data defined in the deposit's event. Following, other validators attest to the request, by submitting a similar transaction, which contains the deposit's information signed by the validator's key.

After a (protocol-defined) threshold of validators attest to a request, the request is executed. Specifically, N wrapped assets are created and assigned to the address α_{dest}.[4]

Remark. It is important that, when a deposit is made on either side of the bridge, the deposit transaction is finalized before the validators approve the corresponding request (for the release of the assets on the other side of the bridge). Alternatively, if a deposit transaction is reverted, e.g., due to a temporary fork in the ledger, the bridge risks violating one of its invariants, which is that the amounts of tokens on either side of the bridge should be always equal. Therefore, the validators should only approve a request, the corresponding deposit of which is k blocks "deep" in the ledger, where k is the safety parameter of each ledger. For example, in Bitcoin k is typically $2-6$, while for Ethereum k is often $14-70$.

Finally, the bridge offers a pausing functionality. The bridge should define a set of invariants, i.e., conditions that should always hold. If one of these invariants is violated, e.g., due to a bug, the bridge's execution should be paused. Such pause allows human intervention, in order to remedy the causes of the failure before restarting the service, thus minimizing potential adverse effects.

Although a mostly similar process is used for sending assets over the bridge in the reverse direction, i.e., depositing wrapped assets on $\mathcal{L}_B$ and retrieving original assets, there exist subtle differences in the two cases, detailed next.

Original → Wrapped Assets The bridge contract on side $\mathcal{L}_A$ handles original ERC-20 tokens. As such, it is the point of entry to the system, since wrapped assets are created only *after* a deposit of original assets is made. Therefore, this contract can handle various different tokens, which all live on $\mathcal{L}_A$.[5] In addition, the bridge contract is only a user of the ERC-20 contract. Therefore, while the wrapped assets circulate on $\mathcal{L}_B$, the original assets are simply held in escrow on the $\mathcal{L}_A$ contract.

Wrapped Assets → Original The bridge contract on $\mathcal{L}_B$ handles wrapped assets. Therefore, it has more control over the creation of the wrapped ERC-20 smart contract (that manages them). Specifically, it is responsible for creating

[4]If a request concerns tokens that have not been bridged before, its execution also creates the ERC-20 contract on $\mathcal{L}_B$, which will maintain the wrapped assets.

[5]For each token, a separate wrapped ERC-20 contract will be created on $\mathcal{L}_B$ during the first deposit request.

the wrapped contract upon executing the first request for an ERC-20 token. The wrapped ERC-20 contract is the same as the ERC-20 contract. Notably, the contract on $\mathcal{L}_B$ makes uses of ERC-20's burning functionality during the process of sending wrapped assets over the bridge back to $\mathcal{L}_A$. Specifically, instead of keeping the sent assets in escrow (as is done by the $\mathcal{L}_A$ contract), the $\mathcal{L}_B$ contract burns the wrapped assets, whose corresponding original assets are released at $\mathcal{L}_A$.

2.2 Fees

Each operation on a distributed ledger, which is done via a transaction, incurs a fee. The fees are paid in the ledger's native currency, e.g., Ethereum transaction (gas) fees are paid in wei.

In the bridging process described above, there exist two types of transactions. First, there exist deposit transactions, which are done by the users who own the assets (original and/or wrapped). In our design, the users are responsible for paying the deposit gas fees. Second, there exist request attestation transactions, which are done by the validators on either side of the bridge. To incentivize the participation of parties as validators, we propose that these fees are also paid - indirectly - by the users of the bridge.

In particular, when a user deposits N assets on either side of the bridge, they receive $N - f$ corresponding assets on the other side. The difference f is the fee paid to the validators. These f assets are kept by the contract. Each validator V has a claim on these fees, which is proportional to the number of attestations V made (over all attestations by all validators combined). We note that these fees are stored in *wrapped assets*. For example, if a user deposits original assets on $\mathcal{L}_A$, the validator fees will be withheld as wrapped assets on $\mathcal{L}_B$; equivalently, if a user deposits N wrapped assets, $N - f$ of them will be burnt (and $N - f$ original assets will be released on the other side) and f of them will be stored as fees.

The exact computation of the amount f per transaction is outside the scope of this document. Nonetheless, it should offer some guarantees. The validators pay fees in two types of tokens (native to the ledgers on either side of the bridge), but receive the reimbursement in a third type (wrapped assets). One resolution to this is to use three *price oracles*, which enable the denomination of these three assets in a single unit of account (e.g., USD). However, using oracles would significantly complicate the design and security of the mechanism. Therefore, an alternative is to trust the bridge's administrator to define, at all times, a sufficient level of fees, which is large enough to cover the validators' expenses.[6]

Protocol fees Some of the fees collected for reimbursing the validators can be awarded to the protocol's designers. The percentage of the split of the fees

[6]Note that this trust assumption is reasonable, since the administrator is anyway trusted for core elements of the bridge's execution (as described below in Section 3).

(between those claimable by the validators and those awarded to the protocol's designers) is also outside the scope of this document.

3 Key Parties

The description of Section 2 outlines a number of parties that play key roles in the bridge's operation. In this section, we consider the role and the trust assumptions that relate to these parties.

3.1 Validators

The primary role in a bridge is that of validators. As described above, validators are responsible for observing both sides of the bridge, listening for deposits and attesting to requests. We note that the sets of validators on either side of the bridge may be different. Therefore, the validators ($\mathbb{V}_{\mathcal{L}_A}$) that listen for deposits on $\mathcal{L}_A$ and attest to requests on $\mathcal{L}_B$ might be different than the ones ($\mathbb{V}_{\mathcal{L}_B}$) responsible for the reverse direction.

Validators control the creation and destruction of assets, as well as who receives assets (on the other side of a deposit). Therefore, the validators should be trusted to behave honestly. In particular, the validators in $\mathbb{V}_{\mathcal{L}_A}$ can create a fake request (on $\mathcal{L}_B$), e.g., which does not correspond to a deposit or which specifies a different address than the one defined in a legitimate deposit on $\mathcal{L}_A$. Therefore, they can create wrapped assets improperly or divert them away from their legitimate owner. Similarly, the validators in $\mathbb{V}_{\mathcal{L}_B}$ can release original assets and divert them to adversarially controlled addresses.

Due to the core role that validators play, it is imperative to carefully choose i) the parties that act as validators and ii) the threshold of attestations necessary for executing a request.

3.2 Administrator

The administrator K is the party responsible for completing a set of tasks, which are elemental to the bridge's secure operation.

First, K parameterizes the validation process. In particular, it chooses the parties who can act as validators, as well as the threshold of attestations needed for a request to be executed (on either side of the bridge).

Second, K defines the fee policy. Specifically, it tunes the level of fees that users pay during deposits, as well as how these fees are split between the set of validators and the protocol's designers.

Third, K is responsible for the pausing process. Specifically, K grants (and removes) pauser roles and is responsible for unpausing a contract. Therefore, K should ensure that, after a pause has occurred, its causes have been resolved and the necessary invariants have been restored.

Finally, K configures possible usability limits on bridge usage, e.g., imposing a minimum or maximum number of tokens per deposit.[7]

Evidently, the administrator plays the most important role in the bridge's operation. If K is corrupted, it can disrupt the entire process of the bridge on all sides. Therefore, it is crucial that its responsibilities are managed by parties that are trusted to operate as securely as possible.

3.3 Pauser

The final special role is that of the pauser node. This node is parameterized by a set of invariants, which are conditions that guarantee the safe and secure operation of the bridge.

The pauser observes all ledgers that participate in the bridge constantly. If, at some point, one of the invariants is violated for a contract on one side of the bridge, the node should pause the bridge. After a contract is paused, no deposits and request executions can be conducted. We note that configuration operations, such as updating the validators' sets or the fee mechanism, can be conducted during a pause period.

The pauser plays an important role, although its control over the contract is limited. In particular, it can disrupt the bridge's operation by pausing the contract without proper cause. This would result in a Denial-of-Service attack, since the bridge would be unusable until it is unpaused. Nonetheless, the pauser cannot gain control or divert the user's assets in any way.

4 Conclusion

In this work we described a mechanism for transferring fungible assets across EVM-compatible distributed ledgers. The core element of the mechanism is "wrapped assets", which are a representation of assets from an originating chain on a destination chain. The process of moving assets across the ledgers is done via a "bridge", which is a pair of smart contracts on either ledgers. The bridge is operated by a set of validators, who listen to asset deposits by users (on one side of the bridge) and are responsible for executing the creation of the corresponding assets (on the other side of the bridge).

4.1 Implementation details

The bridge mechanism has been implemented by Ēnosys, with the implementation available on GitHub.[8] Following, we cover some points of discussion about this particular implementation.

[7]We note that, although the lower threshold has some merit, the upper threshold can be easily bypassed by splitting one's assets across different accounts. Therefore, unless extra assumptions are used (e.g., KYC), it is unclear what guarantees a maximum threshold would provide.

[8]`https://github.com/flrfinance/flr-wraps-contracts`

4.1.1 Validators

The set of validators is split among two committees.[9]

The first committee consists of two members, who act as the development team of the bridge. These entities are Ēnosys and Common Prefix. The threshold for reaching a quorum, that is the minimum number of parties in this committee that need to approve a request before it is executed, is 1 (50%).

The second committee consists of three members, who act as the users and parties of interest for the bridge. These entities are XDC Community, XDC Foundation, and NORTSO. The threshold for reaching a quorum in this committee is 2 (66%).

To execute a request, a quorum on *both* committees is needed. Therefore, at least one attestation from the first committee and at least two attestations from the second committee are needed.

Restrictions The implementation imposes two extra restrictions regarding the validator set. First, if a validator is removed from either of the committees, it cannot be added again on either.[10] Second, the maximum number of members on each committee is 128, so the maximum number of validators is 256 in total (across both commitees).[11]

4.1.2 Administrator

As discussed above, the administrator has complete control over the bridge and can completely disrupt its operation and security. In the implementation, the administrator's key on each side of the bridge is managed by a Gnosis Safe multisig, controlled by Ēnosys.[12]

4.1.3 Deployment Timeline

The tentative deployment timeline of the bridge mechanism is as follows.

The first bridge, which has already been deployed, is between the XDC testnet (Apothem) and the Flare Network testnet (Coston).

The second bridge, which will be deployed after a sufficient testing period, will be between the XDC mainnet and Songbird, Flare Network's "canary" network. At the beginning, this bridge will support a single asset, XDC tokens; therefore, users on XDC will be able to create and use wrapped XDC on Songbird.

[9] `https://github.com/flrfinance/flr-wraps-contracts/blob/master/src/libraries/Multisig.sol`

[10] `https://github.com/flrfinance/flr-wraps-contracts/blob/master/src/libraries/Multisig.sol#L215`

[11] `https://github.com/flrfinance/flr-wraps-contracts/blob/master/src/libraries/Multisig.sol#L44`

[12] `https://gnosis-safe.io`

Following, two bridges are expected to be deployed. The first will be between Ethereum's testnet, Goerli, and Flare Network's testnet, Coston. The second will be between the Ethereum mainnet and Songbird.

Confirmation times The confirmation times that bridge validators need to wait, until a deposit is finalized, are as follows:

- XDC uses an instant finality PoS mechanism, where a committee of validators sign blocks and a block is finalized when $\frac{2}{3}$ of all validators sign it; so, the bridge validators need to wait until a deposit is published in a block with $\frac{2}{3}$ signatures.[13]

- Flare Network uses a variant of Avalanche's Snowman++ algorithm.[14] Although in Flare the election probability depends on FTSO performance (instead of depending only on stake as in Avalanche), the consensus mechanism is the same as the original. Therefore, the confirmation time depends on the parameters k (sample size), α (quorum size) and β (decision threshold). In Flare Network, $k = 20, \alpha = 15, \beta = 15$.[15] Therefore, validators need to wait until a transaction is accepted by 15 of 20 parties in 15 consecutive samples.

- Ethereum (PoS) also employs instant finality, based on a BFT mechanism that requires $\frac{2}{3}$ of all validators to sign blocks. Therefore, again the validators need to wait until a deposit is published in a block that is finalized (meaning signed by $\frac{2}{3}$ of all validators).

4.1.4 Protocol Fees

As discussed above, the fees which are paid by the users are split among the validators and the protocol's development team. In the implementation, the latter are kept only on one side, namely on the side of the bridge that lives on Flare Network.[16] The fees on the other side of the bridges to various ledgers are claimed entirely by the validators.

Specifically, the bridge will charge 1% of the transaction's value in fees. These are split equally between protocol fees, which are given to Ēnosys for the bridge's development, and the validators. At the system's onset, each of the 5 validators described above will have an equal claim on validator fees, which is 0.1% of each transaction's amount.

[13]`https://xinfin.org/xinfin-consensus`

[14]For more details about Avalanche's consensus and the role of the parameters α, β see: `https://docs.avax.network/overview/getting-started/avalanche-consensus`. For a description of Flare's variant see: `https://docs.flare.network/tech/validators/`.

[15]`https://github.com/flare-foundation/go-songbird/blob/222facde7abb18fa7936594205adaa2346faf973/avalanchego/config/flags.go#L264`

[16]`https://github.com/flrfinance/flr-wraps-contracts/blob/master/src/WrapMintBurn.sol#L85`

References

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[W$^+$14] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[WPG$^+$21] Sam M Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William J Knottenbelt. Sok: Decentralized finance (defi). *arXiv preprint arXiv:2101.08778*, 2021.