The long term goals of Lido's integration into Arbitrum protocol are:

- Provide the ability for users to bridge stETH and wstETH tokens into Arbitrum.
- Allow users to stake ETH with Lido from Arbitrum's chain
- Integrate stETH and wstETH tokens in a wide range of protocols on L2: liquidity pools, lending platforms, AMM protocols, etc.
- Keep UX on Arbitrum as close as possible to Ethereum mainnet UX

In this document, the main focus is on the first step: stETH/wstETH bridging into Arbitrum. It's the required step for our integration. Designing and developing staking from the Arbitrum chain will require much more time than just tokens bridging, so we split it into standalone phases and focus on just bridging firstly. We want to launch wstETH bridging in the shortest time, but at the same time, the chosen bridging solution has to provide enough flexibility and extendability to add staking functionality in the future and reach other long-term goals.

Token Bridging Options

Arbitrum provides a "Canonical Bridge" to bridge tokens from Ethereum to Arbitrum and vice versa. Arbitrum's tokens-bridging architecture consists of three types of contracts:

- Asset contracts

: these are the token contracts themselves, i.e., an ERC20 on L1 and its counterpart on Arbitrum.

- Gateways

: pairs of contracts (one on L1, one on L2) that implement a particular type of cross-chain asset bridging.

- Routers

: exactly two contracts (one on L1, one on L2) that route each asset to its designated Gateway.

See the offical Arbitrum's documentation for more detailed info about token bridging.

Arbitrum provides two types of gateways out-of-the-box: [L1\L2ERC20Gateway

](https://developer.offchainlabs.com/docs/bridging_assets#default-standard-bridging) and [L1\L2CustomGateway

](https://developer.offchainlabs.com/docs/bridging_assets#the-arbitrum-generic-custom-gateway). Additionally, a third-party gateway might be developed (for example, Arbitrum Dai Bridge and arbitrum-lpt-bridge) and registered in the Router. It might be helpful when the functionality of default gateways is not enough. Below are considerations on how each of the provided options might be used for bridging wstETH into Arbitrum.

Arbitrum's ERC20 Gateway

Using the default [L1ERC20Gateway

](https://github.com/OffchainLabs/arbitrum/blob/master/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1ERC20Gateway.sol) is one of the simplest options. It doesn't require development and is ready to use at any moment. On the first transfer of wstETH through Arbitrum's [L1GatewayRouter

](https://github.com/OffchainLabs/arbitrum/blob/master/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1GatewayRouter.sol), [L2ERC20Gateway

](https://github.com/OffchainLabs/arbitrum/blob/master/packages/arb-bridge-peripherals/contracts/tokenbridge/arbitrum/gateway/L2ERC20Gateway.sol) (a counterpart gateway of L1ERC20Gateway

on Arbitrum chain) deploys the [BeaconProxy

](https://github.com/OffchainLabs/arbitrum/blob/master/packages/arb-bridge-peripherals/contracts/tokenbridge/libraries/ClonableBeaconProxy.sol) contract on a predefined address (see line 47 of L2ERC20Gateway

contract for address computation logic). The deployed proxy uses Arbitrum's [StandardArbERC20Token

](https://github.com/OffchainLabs/arbitrum/blob/master/packages/arb-bridge-peripherals/contracts/tokenbridge/arbitrum/StandardArbERC20.sol) as an implementation. After the first transfer, the wstETH token can be considered registered in Arbitrum.

Even though Standard ERC20 gateway usage doesn't require development just now, it will significantly increase the time needed to reach other goals of Arbitrum's integration. The main problem is that such an option doesn't allow upgrading the L2 implementation of wstETH. Without upgradeability, future stages of integration will be impossible or very complicated to accomplish. For example, stETH wrapping into wstETH on Arbitrum will be much harder to implement cause Lido will not be able to update implementation to share ERC20Gateway

's minting and burning rights with some exchange Lido's contract. Because of this, such an approach might not be the best option to implement.

- Pros
- No development required
- Fastest solution to launch
- No development required
- Fastest solution to launch
- Cons
- Lack of upgradeability
- No control on Arbitrum's chain token
- Might be hard to reach other Lido goals of integration
- Lack of upgradeability
- No control on Arbitrum's chain token
- Might be hard to reach other Lido goals of integration

Arbitrum's Custom Gateway

The lack of upgradability might be solved using [L1CustomGateway

](https://github.com/OffchainLabs/arbitrum/blob/master/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1CustomGateway.sol). It allows registering which contract to use as a counterpart token on the Arbitrum chain. By default, L1CustomGateway

assumes that registered tokens conform to the ICustomToken

interface. wstETH is not upgradable and doesn't implement such an interface, so it can't be registered directly. Some wrapper/replacement token implementing the required interface might be used instead of wstETH, but additional wrapping or using one more token will worsen the UX. To use wstETH as is, the manual registration via the [forceRegisterTokenToL2()

](https://github.com/OffchainLabs/arbitrum/blob/44730b2d0af7908fe913b2fa85526de925ba6b59/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1CustomGateway.sol#L190) method might be employed. This method is only callable by the Offchain Labs team. It also allows deploying the wstETH token counterpart on Arbitrum behind a proxy to enable future updates (the initial implementation might be StandardArbERC20

with a couple of changes).

Control over the wstETH proxy on Arbitrum's chain allows Lido to upgrade the implementation in the future, but it still doesn't give the control over the bridging process. For example, bridging of stETH token cannot be implemented using either ERC20Gateway

or CustomGateway

due to the token's rebasing mechanics. The only option for direct stETH integration is a custom gateway, which means that wstETH and stETH would be bridged through different gateways and will complicate the process of wstETH <-> stETH swaps on Arbitrum's chain (what is allowed on the Ethereum chain). The problem might be shown with the below example:

- Imagine that we implemented wstETH bridging via CustomGateway

and stETH bridging via a custom LidoGateway

.

- Bob, who bridged 10.0 wstETH into the Arbitrum via CustomGateway

, decided to swap it into 10.1 stETH directly on L2 and withdraw it back to the Ethereum.

- After the dispute period, Bob's withdrawal transaction will fail because LidoGateway

has no stETH.

The same problem is characterized by L1\L2ERC20Gateway

also. A possible solution here: organize exchange via standalone contract, which will transfer collected stETH and wstETH tokens to the Ethereum chain on a daily (or even more often) basis. This will require managing an additional stETH/wstETH buffer on the Ethereum side as well due to the dispute period. But this complexity doesn't manifest when we use the same gateway for wstETH and stETH tokens since our gateway contract holds both tokens, and one can easily swap one for the other on the Ethereum chain without any fees.

- Pros
- Upgradability of wstETH on Arbitrum chain
- Initial solution might use already written contracts for proxy and implementation which might decrease development time.
- Upgradability of wstETH on Arbitrum chain
- Initial solution might use already written contracts for proxy and implementation which might decrease development time.
- Cons
- Development and audit are required
- Integration of stETH requires a standalone gateway which render stETH/wstETH instant swaps on Arbitrum's chain impossible.
- Some time to implement (6-8 weeks)
- Development and audit are required
- Integration of stETH requires a standalone gateway which render stETH/wstETH instant swaps on Arbitrum's chain impossible.
- Some time to implement (6-8 weeks)

Lido's Custom Gateway

L1GatewayRouter

also allows registering a custom gateway for different tokens via the [setGateways()

](https://github.com/OffchainLabs/arbitrum/blob/4577aa73d2c0ac67f74338d0380d3c79a76dba3b/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1GatewayRouter.sol#L208) method, which allows Lido to develop its own gateway for transferring wstETH and stETH tokens into Arbitrum. A third-party gateway must implement the [ITokenGateway

](https://github.com/OffchainLabs/arbitrum/blob/master/packages/arb-bridge-peripherals/contracts/tokenbridge/libraries/gateway/ITokenGateway.sol) interface to be compatible with the L1GatewayRouter

. ITokenGateway

interface doesn't impose strict restrictions on the logic of the gateway and allows Lido to transfer both stETH and wstETH nominated in shares to the counterpart gateway; it eliminates potential problems with rebasable stETH bridging. An additional benefit of such a solution is that both tokens, wstETH and stETH, will be locked in the same contract; as was described in the preivos section, this simplifies the implementation of instant swapping between stETH and wstETH on the Arbitrum chain.

To qualify for the gateway registration, Offchain Labs requests assurances that "once registered and bridged, holders of the representation of the token on L2 wouldn't be exposed to any risks beyond those effectively already born by holders of the token on L1, and that this condition holds even if the L2 token contract code is arbitrarily malicious." A detailed document with a list of assurances should be prepared for stETH and wstETH tokens and shared with the Offchain Labs team.

The minimal gateway implementation will require the development of the following contracts:

- L1LidoGateway

, the gateway implementation for Ethereum chain

- L2LidoGateway

, the gateway implementation of Arbitrum chain

- L2WstETH

, the representation of wstETH on Arbitrum chain

To enable future upgrades, the listed contracts will be deployed behind proxies. The contracts may be based on already implemented gateways for DAI and Livepeer tokens. Such an option provides the most flexibility compared to default gateways, but it also requires the most development time of the three options, the core part of which will be spent on getting an audit for the contracts. In the best scenario, it delays release by a couple of weeks and about a month in a worse case.

The audit step might be excluded from the timeline if we use L1CustomGateway

and L2CustomGateway

as implementation for L1LidoGateway

and L2LidoGateway

without introducing significant changes to their code. The StandardArbERC20Token

, with a couple of changes, might be used as L2WstETH

implementation. These contracts were audited, but the audit reports are not public. If Lido DAO decides that contract changes will not break the audit, the launch of wstETH bridging might happen a couple of weeks earlier than it would do in the scenario of implementing the contrcts from scratch.

- Pros
- Full control over tokens bridging flow
- Upgradability of L2 token and gateway contracts
- The most extensible solution
- Full control over tokens bridging flow
- Upgradability of L2 token and gateway contracts
- The most extensible solution
- Cons
- Development required
- Highly likely requires audit
- Longest time to implement (8-12 weeks including audits)
- Development required
- Highly likely requires audit
- Longest time to implement (8-12 weeks including audits)

Proposed Solution

Based on the pros and cons of the proposed solutions, we suggest using the 3rd option and developing a custom Lido gateway. Custom gateway provides the best foundation for accomplishment of the long-term goals of Arbitrum integration. This option will require more time at the start of the integration but provides the most flexible solution. For example, pausing/unpausing functionality could be added to the gateway, as well as limits for the bridged amount of tokens could be implemented right at the launch. Adding such features excludes the possibility to skip the audit step for speeding up the initial launch. It makes the option to make the initial release fast with the L1/L2CustomGateway

implementation inadvisable. At the same time, it improves the security of user funds (pause the gateway in case of vulnerability, limit the number of total locked tokens). Also, it will simplify future upgrades (pause the gateway to prevent bridging in the process).