Bridging L1 and L2 (Standard Bridge)

Titan supports Standard Bridge for moving assets between Ethereum L1 and Titan L2. Users can send ETH and ERC20 tokens to L2 via a Standard Bridge contract (deposit). Conversely, users can transfer ETH, ERC20 tokens from L2 to L1 via Standard Bridge (withdraw).
To deposit ETH from L1 to L2, you must first use the `depositETH` or `depositETHTo` functions of the `L1StandardBridge` contract to deposit ETH. For ERC20 tokens, use the `depositERC20` or `depositERC20To` functions of the same contract. After the deposit transaction is mined, it takes a few minutes for the assets to be processed and appear in L2.
Withdrawal refers to the movement of assets from L2 to L1. To withdraw ETH and ERC20 tokens from L2, use the `withdraw` and `withdrawTo` functions of the `L2StandardBridge` contract. Unlike L1, withdrawals in L2 are not split between ETH and ERC20, as ETH is treated as a ERC20 token in L2.
Withdrawals from Titan can take up to one week due to the characteristics of Optimistic Rollup and the time required for L2 to validate transactions.
This section will guide you on importing the Titan Contacts package and demonstrate a JavaScript code example for depositing or withdrawing ETH.
The example code can be found here. To learn how to transfer ETH between L1-L2 using the Titan SDK, we recommend running the code yourself.

Setup

- To interact with the Ethereum blockchain, import the `ethers` library. Additionally, obtain the `@tokamak-network/titan-contracts` package to pre-deploy and obtain the ABI and bytecode of the `L1StandardBridge` and `L2StandardBridge` .

```
import { ethers } from "ethers";
import { predeploys } from "@tokamak-network/titan-contracts";

import l1StandardBridgeArtifact from "@tokamak-network/titan-contracts/artifacts/contra
import l2StandardBridgeArtifact from "@tokamak-network/titan-contracts/artifacts/contra
```

- To create a factory instance of each contract, use the artifacts of `L1StandardBridge` and `L2StandardBridge` as parameters, which can be imported via the `@tokamak-network/titan-contracts` package.

```
const factory__L1StandardBridge = new ethers.ContractFactory(
  l1StandardBridgeArtifact.abi,
  l1StandardBridgeArtifact.bytecode
)
const factory__L2StandardBridge = new ethers.ContractFactory(
  l2StandardBridgeArtifact.abi,
  l2StandardBridgeArtifact.bytecode
)
```

- Using the `ethers` library, create a `provider` object and a `wallet` object for L1 and L2. Then create instances of `L1StandardBridge` and `L2StandardBridge` as follows.

1. Create `L2StandardBridge` instance:
   - Use `factory__L2StandardBridge` to leverage methods from a contract factory instance
   - Connect `l2Wallet` to `L2StandardBridge`

- Create an `L2StandardBridge` instance by calling the `attach` function with `predeploys.L2StandardBridge` as the contract address

2. Create `L1StandardBridge` instance:
   - Use `factory__L1StandardBridge` to leverage methods from a contract factory instance
   - Call the `L2StandardBridge` 's `l1TokenBridge()` function to get the L1 Standard Bridge contract address, `L1StandardBridgeAddress` .
   - Connect `l1Wallet` to `L1StandardBridge`
   - Call the `attach` function using `L1StandardBridgeAddress` as the contract address to create an instance of the `L1StandardBridge` .

```
const l1RpcProvider = new ethers.providers.JsonRpcProvider(l1Url)
const l2RpcProvider = new ethers.providers.JsonRpcProvider(l2Url)

const l1Wallet = new ethers.Wallet(key, l1RpcProvider)
const l2Wallet = new ethers.Wallet(key, l2RpcProvider)

const L2StandardBridge = factory__L2StandardBridge
    .connect(l2Wallet)
    .attach(predeploys.L2StandardBridge)
const L1StandardBridgeAddress = await L2StandardBridge.l1TokenBridge()
const L1StandardBridge = factory__L1StandardBridge
  .connect(l1Wallet)
  .attach(L1StandardBridgeAddress)
```

Deposit

- To move ETH from L1 to L2, call the `depositETH` function of the `L1StandardBridge` contract created in the previous step. Ensure that your L1 account has at least the amount of ETH you are depositing, as well as the appropriate gas fee to send the transaction.
- The `depositETH` function has three parameters. The first parameter is the amount of gas to be used for the L2 transaction, while the second parameter is additional data passed in the form of calldata. In the example below, an empty array is used. The third parameter sets the amount of ETH to deposit via the `value` property. This is done by converting the `balance` value to ether using `ethers.utils.parseEther(balance)` .
- After deposit transaction is processed on L1, `receipt` object representing the information for the transaction can be retrieved. You can check the `status` property of the transaction `receipt` object to see if the transaction was processed successfully. If the value of `status` is not 1, it will throw an `Error` object indicating that the transaction has failed.
- The deposit transaction requested from L1 generates a message with `L2StandardBridge` as the destination and forwards it to L2. After passing through `L1CrossDomainMessenger` and `L2CrossDomainMessenger` , the message calls the `finalizeDeposit` function of `L2StandardBridge` to send the amount of ETH deposited to L2.

```
const tx = await L1StandardBridge.depositETH(
  200000, // Gas for L2 transaction
  [],
  {
    value: ethers.utils.parseEther(balance),
  }
)
console.log(`TX Hash: ${tx.hash}`)
```

```
const receipt = await tx.wait()
if (receipt.status !== 1) {
  throw(new Error('transaction is failed'));
}
```

Withdraw

- You can withdraw ETH from L2 to L1 by calling the `withdraw` function of `L2StandardBridge`. (Make sure you have appropriate ETH to pay for the gas fee.)
- The first parameter of the function specifies a L2 token address that specifies the asset to be withdrawn to L1. In our example, we use `predeploys.OVM_ETH`, which uses `OVM_ETH` so that it can be treated as a token. The second parameter sets the amount of ETH to be withdrawn. It is set by converting the `balance` value to ether units using `ethers.utils.parseEther(balance)`. The third parameter is a value representing the withdrawal ownership of the token. In our example, we use 0. The fourth parameter is additional data about the withdrawal, passed as a string. The example uses `'0xFFFF'`.
- We wait for the withdraw transaction to be processed on the L2 and get a `receipt` object representing the receive information for the transaction. You can confirm the `status` property of the transaction's `receipt` object to see if the transaction was processed successfully.
- The withdraw transaction sent by L2 generates a message with `L1StandardBridge` as the destination and forwards it to L1. In an optimal rollup, the transaction and State Root generated on L2 are rolled-up to L1. Once it is confirmed that the State Root has been successfully rolled-up on L1, the message relayer service calls `L1CrossDomainMessenger` to forward the message to L1. Titan supports batch-relay, where multiple messages are delivered in one transaction to lower transaction fees and speed up to relay. The withdrawal is then finalized by calling the `finalizeETHWithdrawal` function of `L1StandardBridge` to send ETH to L1.

```
const tx = await L2StandardBridge.withdraw(
  predeploys.OVM_ETH,
  ethers.utils.parseEther(balance),
  0,
  '0xFFFF'
)

console.log(`TX Hash: ${tx.hash}`)

const receipt = await tx.wait()
if (receipt.status !== 1) {
  throw(new Error('transaction is failed'));
}
```