Bridging L1 and L2 (Cross Domain Messenger)

Titan에서는 Titan SDK를 통해 Cross Domain Messenger를 지원하여 L1와 L2 간에 자산을 자유롭게 이동시킬 수 있습니다. L1의 자산을 L2로 이동하는 것을 deposit이라 하고, L2에서 사용하던 자산을 다시 L1으로 이동시키는 것을 withdraw라고 합니다.
본 문서에서는 Titan SDK를 이용하여 ETH를 deposit/withdraw하는 방법을 Javascript 코드 예제를 통해 알아보겠습니다. Titan SDK를 통해 서비스 개발자는 직접 컨트랙트를 호출하지 않고 L1과 L2 간에 자산을 이동하는 브릿징 기능을 간편하게 구현할 수 있습니다.
실행 가능한 예제 코드는 여기에서 확인할 수 있습니다.

Setup

- Titan SDK 라이브러리를 import하여 라이브러리에서 지원하는 메서드와 타입을 사용할 수 있습니다. ethers 라이브러리는 Ethereum 블록체인과 상호작용하는 데 필요한 기능을 사용하기 위해 import합니다.

```
// Transfers between L1 and L2 using the Titan SDK

const ethers = require("ethers")
const titanSDK = require("@tokamak-network/titan-sdk")
```

- L1 및 L2 네트워크에서 서명을 생성하고 트랜잭션을 전송하기 위해 ethers 라이브러리를 이용하여 provider와 wallet 인스턴스를 생성합니다.
- `l1RpcProvider` : L1 네트워크를 위한 `ethers.providers.JsonRpcProvider` 인스턴스
- `l2RpcProvider` : L2 네트워크를 위한 `ethers.providers.JsonRpcProvider` 인스턴스
- `l1Wallet` : `ethers.Wallet` 을 사용하여 L1 네트워크의 지갑을 생성
- `l2Wallet` : `ethers.Wallet` 을 사용하여 L2 네트워크의 지갑을 생성

```
const l1RpcProvider = new ethers.providers.JsonRpcProvider(l1Url)
const l2RpcProvider = new ethers.providers.JsonRpcProvider(l2Url)
const l1Wallet = new ethers.Wallet(privateKey, l1RpcProvider)
const l2Wallet = new ethers.Wallet(privateKey, l2RpcProvider)
```

- Titan SDK의 BatchCrossChainMessenger 인스턴스를 생성합니다. 생성을 위한 파라미터는 아래와 같습니다. 생성한 인스턴스는 ETH을 전송하기 위해 사용합니다.
  - l1ChainId: L1 네트워크 체인 ID
  - l2ChainId: L2 네트워크 체인 ID
  - l1SignerOrProvider: L1 네트워크 지갑
  - l2SignerOrProvider: L2 네트워크 지갑

```
crossChainMessenger = new titanSDK.BatchCrossChainMessenger({
      l1ChainId: 5,    // Goerli value, 1 for mainnet
      l2ChainId: 5050,  // Goerli value, 55004 for mainnet
      l1SignerOrProvider: l1Signer,
      l2SignerOrProvider: l2Signer,
  })
```

- 여기서 l1ChainId와 l2ChainId는 Titan SDK에서 지원하는 네트워크의 ChainId를 입력해야 합니다. 예를 들어 Titan Mainnet에서 Deposit과 Withdraw를 하기 위해서는 l1ChainId에 이더리움 메인넷의 체인 ID인 1을 입력하고, l2ChainId에 Titan 메인넷의 체인 ID인 55004를 입력합니다. ('Connection information' 섹션 참고)

Deposit

- L1에서 L2로 ETH를 이동하기 위해 deposit 트랜잭션을 요청하는 방법은 아주 간단합니다. Setup 단계에서 생성한 `crossChainMessenger` 인스턴스의 `depositETH()` 를 호출하여 원하는 수량의 ETH를 브릿징할 수 있습니다. (L1에는 deposit하는 수량 이상의 ETH + 트랜잭션 전송을 위한 적정 가스비가 준비되어야 합니다.)
- deposit 트랜잭션 전송 후에는 `wait()` 을 실행합니다. 트랜잭션이 블록에 포함되고 트랜잭션의 receipt이 나올 때까지 대기합니다.

```
// 0.000001 ETH (= 1,000 gwei)
// gwei = 10^9 wei
const response = await crossChainMessenger.depositETH(1000n * gwei)
await response.wait()
```

- deposit 과정에서 내부적으로 L1과 L2에 배포된 컨트랙트들의 함수를 호출하는데 컨트랙트가 다른 컨트랙트에 전달하는 전송 단위를 message라고 합니다.
- `waitForMessageStatus()` 는 첫 번째 파라미터인 message가 두 번째 파라미터에 정의된 message의 status로 변경될 때까지 대기하는 메서드입니다. deposit이 완료되어 L1에서 L2로 ETH가 이동되면 message의 status는 `RELAYED` 로 변경됩니다.
- `response` 가 `RELAYED` 로 변경되었는지 확인하여 deposit이 성공적으로 완료되었는지 체크합니다.

```
// message to wait for RELAYED
await crossChainMessenger.waitForMessageStatus(response, titanSDK.MessageStatus.RELAYED
```

Withdraw

- Titan SDK를 사용하면 L2에서 L1으로 ETH를 손쉽게 이동할 수 있습니다. withdrawETH()를 호출하여 사용자가 원하는 수량만큼 ETH를 브릿징할 수 있습니다. (L2에는 withdraw하는 수량 이상의 ETH + 트랜잭션 전송을 위한 적정 가스비가 준비되어야 합니다.)

```
// 0.000001 ETH
const response = await crossChainMessenger.withdrawETH(1000n * gwei)
await response.wait()
```

- L2에서 withdraw 트랜잭션을 요청하면 생성된 L2 트랜잭션들은 L2에서 L1으로 트랜잭션과 State Root를 롤업하는 sequencer와 proposer에 의해 롤업됩니다. `waifForMessageStatus()` 를 호출하여 message의 status가 `READY_FOR_RELAY` 로 변경될 때까지 대기합니다. proposer에 의해 State Root가 L1으로 성공적으로 롤업되면 message의 status가 `READY_FOR_RELAY` 로 변경됩니다. 메인넷 기준 Optimistic Rollup에서 fraud proof를 위한 검증 기간으로 7일을 정의하고 있기 때문에 `READY_FOR_RELAY` 상태로 변경될 때까지 7일이 소요됩니다.

```
await crossChainMessenger.waitForMessageStatus(response, titanSDK.MessageStatus.READY_F
```

- `finalizeBatchMessage()` 를 호출하여 L2 → L1으로 릴레이 트랜잭션을 요청합니다. 메서드의 파라미터로는 message의 배열이 입력되어야 합니다. `finalizeBatchMessage()` 은 배열에 저장된 message들을 하나의 트랜잭션에 담아서 한꺼번에 릴레이합니다.
- withdraw가 성공적으로 완료되어 L2에서 L1로 원하는 수량의 ETH가 이동되면 message의 status는 `RELAYED` 로 변경됩니다.