Cross-Chain Bridges

Learn to support multi-chain dApp use cases with cross-chain bridges.

Cross-chain bridges enable developers to build dApps, decentralized exchanges, and payment protocols using assets native to other blockchains.

There are currently two options for cross-chain bridges between Filecoin and other blockchains,Axelar andCeler . This cookbook will focus on the use of Celer since it is available on both Calibration testnet and Mainnet, while Axelar is currently only available on Mainnet.

Token Transfers with cBridge

cBridge is a cross-chain asset transfer solution that does not require upfront liquidity.

Ingredients

- Celer Documentation
- Celer Tutorial
- Celer cBridge SDK
- A fulltutorial
- on how to develop a smart contract as a liquidity pool
-

Instructions

1. Sender sendstransferOut
2. tx on the source chain.
3.

```

Copy / *@devtransfer sets up a new outbound transfer with hash time lock./* functiontransferOut( address_bridge, address_token, uint256_amount, bytes32_hashlock, uint64_timelock, uint64_dstChainId, address_dstAddress )external{ bytes32transferId=_transfer(_bridge,_token,_amount,_hashlock,_timelock); emitLogNewTransferOut( transferId, msg.sender, _bridge, _token, _amount, _hashlock, _timelock, _dstChainId, _dstAddress ); }

```

1. Bridge node sendstransferIn
2. tx on the destination chain, using the samehashlock
3. set by the sender.
4.

```

Copy / *@devtransfer sets up a new inbound transfer with hash time lock./* functiontransferIn( address_dstAddress, address_token, uint256_amount, bytes32_hashlock, uint64_timelock, uint64_srcChainId, bytes32_srcTransferId )external{ bytes32transferId=_transfer(_dstAddress,_token,_amount,_hashlock,_timelock); emitLogNewTransferIn( transferId, msg.sender, _dstAddress, _token, _amount, _hashlock, _timelock, _srcChainId, _srcTransferId ); }

```

1. Senderconfirms
2. the transfer on the source chain.
3. Bridge nodeconfirms
4. the transfer on the destination chain.
5.

```

Copy / *@devconfirm a transfer. **@param_transferId Id of pending transfer. *@param_preimage key for the hashlock*/ functionconfirm(bytes32_transferId,bytes32_preimage)external{ Transfermemoryt=transfers[_transferId];

require(t.status==TransferStatus.Pending,"not pending transfer");
require(t.hashlock==keccak256(abi.encodePacked(_preimage)),"incorrect preimage");

transfers[_transferId].status=TransferStatus.Confirmed;

IERC20(t.token).safeTransfer(t.receiver,t.amount); emitLogTransferConfirmed(_transferId,_preimage); }
```

```

The contract addresses for Celer are as follows:

Name Mainnet Calibration wFIL 0x60E1773636CF5E4A227d9AC24F20fEca034ee25A USDC 0x2421db204968A367CC2C866CD057fA754Cb84EdF 0xf5C6825015280CdfD0b56903F9F8B5A2233476F5 USDT 0x422849b355039bc58f2780cc4854919fc9cfaf94 0x7d43AABC515C356145049227CeE54B608342c0ad WBTC 0x592786e04c47844aa3b343b19ef2f50a255a477f 0x265B25e22bcd7f10a5bD6E6410F10537Cc7567e8 WETH 0x522b61755b5ff8176b2931da7bf1a5f9414eb710 0x5471ea8f739dd37E9B81Be9c5c77754D8AA953E4 For further details on cBridge transfers, see the Celer created Github repoHERE .

Interchain Messaging

Celer also enables general message passing between chains. Below is sample code showing how one party can send a message to a counterparty on a different blockchain.

Ingredients

- Inter-chain Messaging
-

Instructions

1. Someone looking to send a message to a wallet on another chain sends that message using the the functionsendMessage()
2. .
3.

```

Copy // called by user on source chain to send cross-chain messages functionsendMessage(address_dstContract, uint64_dstChainId, bytescalldata_message)externalpayable{ bytesmemorymessage=abi.encode(msg.sender,_message); sendMessage(_dstContract,_dstChainId,message,msg.value); }

```

1. The functionexecuteMessage()
2. is used by the intended recipient in the destination chain to receive and emit the message.
3.

```

Copy // called by MessageBus on destination chain to receive cross-chain messages functionexecuteMessage(address_srcContract, uint64_srcChainId, bytescalldata_message, address// executor)externalpayableoverrideonlyMessageBusreturns(ExecutionStatus) { (addresssender,bytesmemorymessage)=abi.decode((_message), (address,bytes)); emitMessageReceived(_srcContract,_srcChainId,sender,message); returnExecutionStatus.Success; }

```

The MessageBus contract addresses are below:

Name Mainnet Calibration MessageBus 0x6ff2130fbdd2837b0c92d7f56f6c017642d84f66 0xd5818D039A702DdccfD11A900A40B3dc6DA03CEc For more information on cross-chain messaging, see the Celer documentationhere .

A note on Finality with Celer

Note that there is an expected finality period when conducting inter-chain messaging with Celer. See details on Filecoin's finalityhere . There are two incoming improvements that developers can follow for the latest developments:

1. FIP86 for fast finality in Filecoin
2. Ready-to-use EC finality calculator
3.

Learn more about cross-chain bridges and which bridges are available on which networks in the Filecoin Docshere .

Previous Oracles Next Decentralized Database

Last updated21 days ago
```