Authentication

Authentication is a critical concept to understand when building xApps. In the context of smart contracts, an authenticated call is one that passes permissioning constraints set by the protocol developer. In most cases this manifests as a modifier that allows a certain set of addresses to call specific smart contract functions - in other words, we are talking about access control.

For example:

- Uniswap'sswap
- function
- isunauthenticated
- because it is a public function that can be called by anyone.
- Uniswap'ssetFeeProtocol
- function
- isauthenticated
- because it uses theonlyFactoryOwner
- modifier that prevents anyone but the owner of the contract factory from calling it. You can read more about this atOpenZeppelin's Ownable contracts
- .
-

The major impact that an authenticated call can have on users is latency. Authenticated calls are a crucial feature for xApps but in order to validate data transferred between chains, Connext must allowsome time to elapse before accepting messages as authenticated on destination chains. This latency is both a drawback and a security mechanism of optimistic bridges.

Checking Origin Data

Suppose a target contract on the destination domain has a function that should only be callable by a specific source contract on a specific origin domain.

A custom modifier likeonlySource below can conduct all the necessary checks to uphold this permissioning constraint.

```
```

Copy contractTargetisIXReceiver{ /*@noticeA modifier for authenticated calls. * This is an important security consideration. If the target contract * function should be authenticated, it must check three things: * 1) The originating call comes from the expected origin domain. * 2) The originating call comes from the expected source contract. * 3) The call to this contract comes from Connext.* / modifieronlySource(address_originSender,uint32_origin) { require(origin==&& _originSender==&& msg.sender==, "Expected source contract on origin domain called by Connext");; }

functionxReceive(bytes32_transferId, uint256_amount, address_asset, address_originSender, uint32_origin, bytesmemory_callData)externalonlySource(_originSender,_origin)returns(bytesmemory) { // Do stuff that requires authentication } }

```
```

You can find a full example of this atAuthenticated Greeter.

Previous Tracking xCalls Next Handling Failed xCalls Last updated9 months ago On this page Edit on GitHub