Across Settlement Integration

The future of interoperability is intents and Across Settlement is the only production-ready, modular settlement layer built to facilitate cross-chain intents. The Across Bridge and Across+ products build on top of Across Settlement to offer bridging to users and bridge abstraction to dApp developers. With Across Settlement, any other application or source of cross-chain intent order flow can leverage Across Settlement to enable the fastest speeds and lowest fees for their interoperability use-case.

Integrating Across Settlement into Your Application

This guide contains an examples for how to build on top of Across Settlement If you have further questions or suggestions for this guide, please send a message to the

developer-questions

channel in the Across Discord .

Across Settlement Example

Across Settlement is a very flexible system, so this example won't capture all that you can do. However, it should illustrate the types of integrations that are possible.

Architecture

The example application we're going to build has the following architecture: 1. 1. 2. User asks for a quote on a cross-chain swap from an offchain RFQ system. 3. 2. 4. Market-maker-relayers bid on the opportunity to fill the user. 5. 3. 6. The user signs the winning bid as a Permit2 order. 7. 4. 8. The Permit2 order is passed to the winning market-maker-relayer. 9. 5. 10. The market-maker-relayer sends a transaction where this order is put onchain. 11. 6. 12. In that transaction, the user's funds are pulled via Permit2. A bond is submitted by the market-maker-relayer in the transaction as well in the same currency as the user's funds. 13. 7. 14. An Across deposit is created. The bond + user funds are set as the input amount. The market-maker-relayer is set as the exclusive relayer, so they are the only relayer who can submit the relay on the destination. This tutorial will not implement the offchain components, but it will show a sample contract implementation for the contract that processes the Permit2 order.

Sample Implementation

Note: the implementation below has not been vetted, validated, or optimized in any way. It is purely meant as an illustrative example. Use this sample code at your own risk. See the comments in the code to understand what each portion of the code is doing. import

{ IPermit2 }

from

"permit2/src/interfaces/IPermit2.sol" ; import

{ IERC20 }

from

"openzeppelin-contracts/contracts/token/ERC20/IERC20.sol" ; import

{ SafeERC20 }

from

"openzeppelin-contracts/contracts/token/ERC20/utils/SafeTransferFrom.sol" ; // Across SpokePool Interface interface

AcrossV3SpokePool

{ function

depositV3 (address depositor , address recipient , address inputToken , address outputToken , uint256 inputAmount , uint256 outputAmount , uint256 destinationChainId , address exclusiveRelayer , uint32 quoteTimestamp , uint32 fillDeadline , uint32 exclusivityDeadline , bytes

calldata message)

external ; } // Sample cross-chain order struct

CrossChainOrder

{ address rfqContract ; address swapper ; address relayer ; address recipient ; address inputToken ; address outputToken ; uint256 inputAmount ; uint256 relayerBondAmount ; uint256 outputAmount ; uint256 destinationChainId ; uint256 nonce ; uint256 deadline ; } // A typestring is required for Permit2 to be able to process the order as witness data. string

constant ORDER_WITNESS_TYPESTRING =

"CrossChainOrder witness)CrossChainOrder(address rfqContract,address swapper,address relayer,address recipient,address inputToken,address outputToken,uint256 inputAmount,uint256 relayerBondAmount,uint256 outputAmount,uint256 destinationChainId,uint256 nonce,uint256 deadline)" ; contract

RFQSwapper

{ using

SafeERC20

for IERC20 ; IPermit2 public immutable permit2 ; AcrossV3SpokePool public immutable spokePool ;

constructor (address _permit2 ,

address _spokePool)

{ permit2 =

Permit2 (_permit2); spokePool =

AcrossV3SpokePool (_spokePool); }

// The initiate function: // 1. Takes in an order and swapper signature. // 2. Verifies the signature and order validity. // 3. Passes along the tokens and instructions to the Across SpokePool // where the order will be filled, that fill verified, and then settled. function

initiate (CrossChainOrder memory order ,

bytes

calldata signature)

{ // This code is the basic process for validating and pulling in tokens // for a Permit2-based order. // It's somewhat involved, but it does most of the work in validating the // order and signature. permit2 . permitWitnessTransferFrom (// PermitTransferFrom struct init IPermit2 . PermitTransferFrom ({ permitted : IPermit2 . TokenPermissions ({ token : order . inputToken , amount : order . inputAmount }), nonce : order . nonce , deadline : order . deadline }), // SignatureTransferDetails struct init IPermit2 . SignatureTransferDetails ({ to :

address (this), requestedAmount : order . inputAmount }), order . swapper , keccak256 (abi . encode (order)); ORDER_WITNESS_TYPESTRING , signature);

// Pull in the bond from the msg.sender. IERC20 (order . inputToken). safeTransferFrom (msg . sender , order . relayerBondAmount);

// Full input amount for Across's purposes both the user amount // and the bond amount. // In the case that the relay is filled correctly, the relayer // gets the full input amount (including the bond). // In the case that the depositor is refunded, they receive the bond // as compensation for the relayer's failure to fill. uint256 amount = order . inputAmount + order . relayerBondAmount ;

// Now that all the tokens are in this contract, Across contract needs // to be approved to pull the tokens from here. IERC20 (order . inputToken). safeIncreaseAllowance (address (spokePool), amount);

// Fill deadline is arbitrarily set to 1 hour after initiation. uint256 fillDeadline = block . timestamp +

3600 ;

// Call deposit to pass the order off to Across Settlement. spokePool . deposit (order . swapper , order . recipient , order . inputToken , order . outputToken , order . inputAmount , order . outputAmount , order . destinationChainId , order . relayer , block . timestamp , fillDeadline ,

// 1 hour deadline fillDeadline ,

// Exclusivity for the entire fill period ""

// No message); } }

Conclusion

As is shown in the example above, Across Settlement can be used as a tool to verify cross-chain intent fulfillment in many different setups. The primary advantage is that the integrating protocol can abstract away the complexity of the verification and fulfillment, so they don't have to deal with the security, timing, and asynchrony challenges of building a cross-chain system. Across Settlement provides additional advantages over naive cross-chain settlement systems, which ultimately lead to better execution of intent fulfillment for users and relayers: 1. 1. 2. Aggregated and Optimistic Verification: The Across Settlement system aggregates valid fills events off-chain to create a repayment bundle, which is then optimistically verified by 3. UMA's Optimistic Oracle 4. . This verification and repayment mechanism scales gas cost of repayment at O(1) instead of O(N) with the number of fills. This offers an order of magnitude in gas savings vs. other approaches and ultimately leads to better pricing for users and more profit for relayers. 5. 2. 6. Relayer Cross-chain Management: With Across' settlement architecture, repayment is made on the relayer's chain of choice, reducing overhead and complexity of managing cross-chain positions. This lowers costs for relayers, enabling better pricing and execution for end-users. It is enabled by Across' Hub and Spoke model, where passive LPs extend loans to relayers for taking on time-value risk as funds are rebalanced through canonical bridges by the protocol. Integration Guides -Previous Across+ Integration Next- Reference API Last modified17d ago On this page Integrating Across Settlement into Your Application Across Settlement Example