Sealed Bid Auction Developer Tutorial

Overview

A first-price sealed-bid auction, also known as a "blind auction", is a type of auction in which all bidders simultaneously submit sealed bids so that no bidder knows the bid of any other participant. The highest bidder pays the price that was submitted. In this tutorial you will learnhow to create a cross-chain sealed bid auction dApp with encrypted bids using SecretPath .

See a live demohere , configured for Sepolia testnet!(To use the demo, make sure Sepolia testnet is added to your Metamask wallet). See the fullstack frontend code implementationhere . You will start by configuring your developer environment and then learn how to use SecretPath to enable cross-cross-chain encryption and decryption,using Secret Network as a Confidential Privacy Layer (CPL) for the EVM .

Getting Started

To get started, clone the Sealed Bid Auction repo:

```

Copy gitclonehttps://github.com/writersblockchain/sealed-bid-auctions

```

EVM Prerequisites

1. Add Sepolia testnet to Metamask.
2. Fund
3. your Sepolia wallet.
4.

Secret Network Prerequisites

1. Add Secret Network testnet to Keplr
2. .
3. Fund your Secret testnet wallet
4. .
5.

Uploading Sealed Bid Contract

cd intosealed-bid-auction/secret-contract

```

Copy cdsecret-contract

```

Install the node dependencies

```

Copy npminstall

```

Update theenv file with your Secret Network wallet mnemonic, and rename it ".env" instead of ".env.example"

Compile the contract

```

Copy makebuild-mainnet

```

cd intosecret-contract/node:

```

Copy cdnode

```

Set SecretPath parameters:

Open upload.js and configure the SecretPathgatewayAddress ,gatewayHash , andgatewayPublicKey :

```

Copy constgatewayAddress="secret10ex7r7c4y704xyu086lf74ymhrqhypayfk7fkj";

constgatewayHash= "012dd8efab9526dec294b6898c812ef6f6ad853e32172788f54ef3c305c1ecc5";

constgatewayPublicKey= "0x046d0aac3ef10e69055e934ca899f508ba516832dc74aa4ed4d741052ed5a568774d99d3bfed641a7935ae73aac8e34938db747c2f0e8b2aa95c25d069a575cc8b";

```

gatewayAddress, gatewayHash, and gatewayPublicKey are needed for instantiating contracts that utilize SecretPath and can be found in the docshere . You will always use these same 3 parameters for instantiating a SecretPath-compatible contract on testnet. Upload and instantiate the contract:

```

Copy nodeupload

```

Upon successful upload and instantiation, add the contract codehash and address to yourenv file .

List an Auction Item

Now that you've instantiated a sealed bid contract on Secret Network, it's time to create your first auction item with SecretPath!

cd intosealed-bid-auctions/evm-contract :

```

Copy cdsealed-bid-auctions/evm-contract

```

Install the dependencies

```

Copy npmi

```

Configure env

Configure theenv file with your sealed bid auction contract address and codehash, and rename it ".env" instead of ".env.example".

Configure SecretPath

Openscripts/create_auction.js and navigate toline 44 , thepublicClientAddress . This is the SecretPath gateway address forSepolia testnet .

If you wanted to send messages on another chain, such as Base or Polygon, you would simply update this publicClientAddress with the corresponding address found here . Similarly, there is a SecretPath gateway encryption key, which is online 63. This is used for ChaCha20-Poly1305 Payload encryption and can be found in the docshere.

If you wanted to do this for mainnet, you would simply use the mainnet encryption key. Next, configure the auctionname ,description, andend_time to your liking (end_time is the amount of minutes that the auction will be live for), and note thehandle variable, which is the function that is actually being called in the Secret contract that you deployed. You are executing thecreate_auction_item handle, which executes thecreate_auction-item function in your sealed bid contract.

Now that you have all of your SecretPath code configured, execute the SecretPath Sepolia public gateway contract to send your auction item to the Secret contract:

```

Copy npxhardhatrunscripts/create_auction.js--networksepolia

```

Upon successful execution, info about your SecretPath payload will be returned:

```

Copy PayloadHash:0x6a822118ea803fe9274c502d354dfd6b24e99f9a67b8b4b11032649dc4b82da1 Payload Signature:
0x77e3a0fd7bb8ea7d96889ed82521672ee0a7e0c5cb81cdc2187e7253407e8f136204d578f13a832dcacb1b2509118114b4aca0635984bb1d7d673579572cf9261b Recovered public key:
0x0423d8d8b518902cd6b0da592af0424719c355a724687cb74d96bd1171eb148edb87f3e9ca67f9ccecde109333461162af4dc09b33604c7e82242852a7142878ec
Verifythismatchestheuseraddress:0x49e01eb08bBF0696Ed0df8cD894906f7Da635929 _userAddress:0x49e01eb08bBF0696Ed0df8cD894906f7Da635929
_routingInfo:secret1xw2ge736z7la2fwuwwh89edwcxks4dv3qf2mg8 _payloadHash:0x6a822118ea803fe9274c502d354dfd6b24e99f9a67b8b4b11032649dc4b82da1 _info:
{"user_key":"0x026c4af0a833ed26f82058d882a0e18114ac398eebdc05b206d11a9060c075026c","user_pubkey":"0x0423d8d8b518902cd6b0da592af0424719c355a724687cb74d96bd1171eb148edb87f
3","handle":"create_bid","nonce":"0x0d0075c1a22be720ad003eba","payload":"0x0b47475944f9a0c966f9b8be7f779f291c31eb11a78132a19fb23aef1b97a92b189e938b5d1975d81836f2ff0aa89e0ade4b4
_callbackAddress:0x3879e146140b627a5c858a08e507b171d9e43139, _callbackSelector:0x373d450c, _callbackGasLimit:300000
Transactionsent!Hash:0xae5dd78f381b67e470a70eaf757c306a1ee605f145007cd7a6e4f4cb8d56be4b Transactionconfirmed!BlockNumber:5683035

```

Bid On Auction Item

Now it's time to place an encrypted bid on your listed auction item. Openbid.js and adjust theamount that you want to bid as well as theindex of the auction item.

Note that the sealed bid contract is designed so that each auction item has an ascending index number starting with 1. So the first auction item you list is index 1, the second is index 2, and so on. Once you have set your bid, execute the bid function:

```

Copy npx hardhat run scripts/bid.js --network sepolia

```

Upon successful execution, info about your SecretPath payload will be returned. Now let's query your auction item and bids with secret.js.

Querying Auction Items and Bids

cd intosealed-bid-auctions/secret-contract/node :

```

Copy cdsealed-bid-auctions/secret-contract/node

```

Make sure you have added your Sealed bid contract address and codehash to your env file, and then query the auction item withnode query_auction :

```

Copy nodequery_auction

```

If your auction item was submitted successfully, it should be returned like so:

```

Copy { name:'auction item #1', description:'this is the 1st auction item', end_time:4397696, message:'Retrieved value successfully' }

```

NOTE:end_time is converted from minutes to Secret Network block height in the sealed bid auction contract 😎 Now, query the bids by runningnode query_bid :

```

Copy node query_bid

```

If the bidding is still open, it will return the message:

```

Copy { message:'Bidding is open'}

```

If the bidding is closed, it will return the highest bid:

```

Copy { message:'Bidding is closed. The highest bid is: 300'}

```

This is programmed in theretrieve_bids_query function of the Sealed Bid contract and can be adjusted to your liking 😊

NOTE: Be sure toupdate the index of the query for subsequent auction item queries

Conclusion

Congrats! You deployed your very own sealed bid auction contract on Secret Network and used SecretPath to send cross-chain encrypted bids on Sepolia testnet. You now have all of the tools you need to start building your own cross-chain SecretPath contracts on the EVM

If you have any questions or run into any issues, post them on theSecret Developer Discord and somebody will assist you shortly.

Last updated6 days ago On this page * Overview * Getting Started * EVM Prerequisites * Secret Network Prerequisites * Uploading Sealed Bid Contract * List an Auction Item * Bid On Auction Item * Querying Auction Items and Bids * Conclusion

Was this helpful? Edit on GitHub Export as PDF