Confidential Voting Developer Tutorial with ECDH

Learn how to use Secret Network smart contracts to encrypt and decrypt votes on Polygon testnet.

Intro

In this tutorial you will learnhow to encrypt and decrypt votes on the EVM with Secret Network smart contracts so that you can build confidential voting on any EVM chain of your choosing. You will be working with two smart contracts:

1. EVM smart contract
2. deployed on Polygon testnet (voting contract)
3. Secret Network smart contract
4. deployed on Secret testnet (decryption contract)
5.

The EVM contract stores encrypted proposals and votes, and the Secret contract decrypts the stored votes and reveals them in a query.

See alive demo here , configured for Polygon testnet!(To use the demo, make sure Polygon testnet is added to your Metamask wallet) You will start by configuring your developer environment and then learn how to generate cryptographic keys in a Secret Network smart contract which you will use to encrypt votes on the EVM.

Getting Started

To get started, clone theSecret Labs examples repo :

```
Copy gitclonehttps://github.com/scrtlabs/examples.git
```

EVM Prerequisites

1. Add Polygon Mumbai testnet to Metamask
2. .
3. Fund your Mumbai wallet
4. .
5.

Secret Network Prerequisites

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

Configuring Environment Variables

cd into examples/evm-confidential-voting/polygon:

```
Copy cdexamples/evm-confidential-voting/polygon
```

Install the node dependencies:

```
Copy npminstall
```

Update theenv file with your Secret Network wallet mnemonic, EVM wallet private key, andInfura API key:

Make sure your Infura API key is configured for Polygon Matic testnet 😎 Next, generate encryption keys for your EVM contract and automatically add them to yourenv file by runningcreate_keys.js :

```
Copy npxhardhat--networkpolygonrun./scripts/create_keys.js
```

Now you are ready to upload the smart contracts

Upload and Instantiate Secret contract

cd into examples/evm-confidential-voting/secret-network:

```
Copy cdexamples/evm-confidential-voting/secret-network
```

Compile the Secret Network smart contract:

```
Copy makebuild-mainnet
```

If you are on a Mac and run into compilation error:

error occurred: Command “clang”

Make sure you have thelatest version of Xcode installed and then update your clang path by running the following in your terminal:

cargo clean

AR=/opt/homebrew/opt/llvm/bin/llvm-ar CC=/opt/homebrew/opt/llvm/bin/clang cargo build --release --target wasm32-unknown-unknown Seehere for instructions on updating your clang path. cd into examples/evm-confidential-voting/secret-network/node

```
Copy cdexamples/evm-confidential-voting/secret-network/node
```

Install the node dependencies:

```
```

Copy npminstall

```

Upload the Secret Network smart contract:

```

Copy nodeindex

```

Upon successful upload acodeHash and contractaddress is returned:

```

Copy Startingdeployment… codeId:3226 Contracthash:4fb0433133d3e9441790ab713ad8000bb99c3894a36b679f355ffaea052035b9 Instantiatingcontract… contractaddress:secret1lft908ws8h034zpa6y2gsq2shpsksekl05gqgq

```

Update theenv file with yourcodeHash and contractaddress :

Execute Secret Network Smart Contract

Now that your Secret Network smart contract is instantiated, you can execute the contractto generate encryption keys as well as decrypt encrypted messages . To learn more about the encryption schema, see the EVM encryption docshere .

Create Keys

To create encryption keys, runnode create_keys :

```

Copy nodecreate_keys

```

After you generate your keys successfully, query your public key:

```

Copy nodeget_keys

```

Which returns your public key as astring :

```

Copy 2,251,34,75,188,184,127,245,254,38,103,132,60,248,107,222,239,201,55,224,56,34,139,127,66,213,21,19,126,68,113,76,233

```

Add the public_key to yourenv file:

```

Copy SECRET_PUBLIC_KEY=2,251,34,75,188,184,127,245,254,38,103,132,60,248,107,222,239,201,55,224,56,34,139,127,66,213,21,19,126,68,113,76,233

```

Now it's time to upload a Voting contract to the EVM, which you will use tostore encrypted votes that can only be decrypted by your Secret Network smart contract.

Upload and Instantiate Polygon Smart Contract

cd into examples/evm-confidential-voting/polygon:

```

Copy cdevm-confidential-voting/polygon

```

Compile your Solidity smart contract:

```

Copy npxhardhatcompile

```

Once the contract is compiled successfully, upload the contract to Polygon testnet:

```

Copy npxhardhatrunscripts/deploy.js--networkpolygon

```

Note the contract address:

```

Copy PrivateVotingdeployedto:0x90c6C32994c622f3882579C76C4b4c41022da494

```

Add the Polygon testnet contract address to yourenv file:

```

Copy CONTRACT_ADDRESS="0x90c6C32994c622f3882579C76C4b4c41022da494"

```

Execute Polygon Testnet Smart Contract

Now that your Polygon smart contract is instantiated, you can execute the contract tocreate voting proposals as well as vote on existing proposals . You can review the solidity contracthere .

Create Voting Proposal

To create aproposal , you must include aproposal description (a "yes" or "no" question) as well as aquorum number, which is the number of unique wallet addresses required to vote on the proposal before it closes.

For testing purposes, setquorum to 1 unless you want to test with multiple wallet addresses ```

Copy functioncreateProposal(stringmemorydescription,uintquorum)externalreturns(uintproposalId) { proposalId=nextProposalId; nextProposalId++; Proposalstorageproposal=proposals[proposalId];
```

proposal.id=proposalId; proposal.description=description; proposal.quorum=quorum; emitProposalCreated(proposalId,description); }

```

Opencreate_proposal.js and update theproposal_description to a "yes" or "no" question of your choice:

```
Copy constproposal_description="Do you love Secret?";
```

Then runcreate_proposal.js :

```
Copy npxhardhat--networkpolygonrun./scripts/create_proposal.js
```

Atransaction hash will be returned upon successful execution:

```
Copy Transactionhash:0x1b26f860328a4f01236dac49cd20cfe4a06a80826514bb7a46a7ec890886ca4c CreateProposalfunctionexecutedsuccessfully!
```

You can query the proposal by runningquery_by_proposal_id :

```
Copy npxhardhat--networkpolygonrun./scripts/query_by_proposal_id.js
```

Be sure to update theproposalId inquery_by_proposal_id.js with theproposalId you want to query! Which returns your proposal:

```
Copy FetchedProposal:{id:1,description:'Do you love Secret?',quorum:1,voteCount:0}
```

Each time you create a proposal, theproposalId is incremented by 1. Your firstproposalId is 1, your nextproposalId will be 2, and so on.

Vote on Proposal

Now it's time to vote on the proposal you created. Openvote.js and update your proposal answer to either "yes" or "no" in themsg object:

```
Copy letmsg={ answer:"yes", proposal_id:proposal.id, proposal_description:proposal.description, salt:Math.random(), };
```

proposal.id andproposal.description will match the proposal info you input forgetProposalById.

This means that each time you vote, you need to make sure you update the proposal_id number that you pass to getProposalById() so that it matches the proposal you want to vote on! ```

Copy letproposal=awaitgetProposalById(1);

```

Once you have updated yourvote andproposalId , execute the vote script:

```
Copy npxhardhat--networkpolygonrun./scripts/vote.js
```

Your encrypted data and transaction hash are returned upon successful execution:

```
Copy Encrypteddata:Uint8Array(120) [ 115,69,78,152,84,64,134,83,152,110,15,162, 90,131,84,73,128,158,159,39,103,8,131,246, 61,95,230,131,220,79,25,68,203,174,180,168,
244,71,125,190,46,173,207,217,150,249,150,223, 69,229,64,98,255,145,141,136,158,181,97,137, 148,71,25,213,184,165,116,224,80,201,138,211, 3,112,237,103,209,77,200,23,52,178,220,147,
143,153,120,151,74,140,137,174,86,3,38,200, 64,197,168,165, ...20moreitems ] Transactionhash:0xd69235d34c0c326cf224661264035feec453eacd5749cbabdf7a018b6285d4f2
votefunctionexecutedsuccessfully!
```

Decrypt Votes

Now it's time to decrypt your vote! First, query that the vote was successfully added to the proposal by runningquery_proposal_votes.js :

Be sure to update theproposalId with the proposal you want to query. ```

Copy npxhardhat--networkpolygonrun./scripts/query_proposal_votes.js

```

query_proposal_vote returns your encrypted vote for the suppliedproposalId :

```
Copy [
'0x73454e9854408653986e0fa25a835449809e9f27670883f63d5fe683dc4f1944cbaeb4a8f4477dbe2eadcfd996f996df45e54062ff918d889eb56189944719d5b8a574e050c98ad30370ed67d14dc81734b2
]
```

Rundecrypt.js to decrypt the vote:

```
Copy npxhardhat--networkpolygonrun./scripts/decrypt.js
```

Indecrypt.js , update theproposalId with the proposal you want to query. Which returns your decrypted vote:

```
Copy { votes:[ '{"answer":"yes","proposal_id":1,"proposal_description":"Do you love Secret?","salt":0.20849165534651148}' ] }
```
```

Conclusion

Congrats! You have now deployed smart contracts on Polygon and Secret Network and implemented private cross-chain voting. If you have any questions or run into any issues, post them on theSecret Developer Discord and somebody will assist you shortly.

Last updated27 days ago On this page