# Entropy

Pyth Entropy allows developers to quickly and easily generate secure

random numbers on the blockchain. Entropy\'s rapid response time allows

developers to build applications such as NFT mints and games with

responsive UX. Entropy also provides [strong security

guarantees](https://docs.pyth.network/entropy/protocol-design) to ensure

that both users and application developers can trust that the results

are random.

Pyth Entropy is currently available on several [EVM

networks](https://docs.pyth.network/entropy/contract-addresses). If you

would like a deployment on another network, please [ask in

Discord[(opens in a new

tab)]{.underline}](https://discord.gg/invite/PythNetwork).

## Getting Started

Using Pyth Entropy is permissionless and developers can integrate in a

few minutes. Please see [How to Generate Random Numbers Using Pyth

Entropy](https://docs.pyth.network/entropy/generate-random-numbers) to

start integrating Pyth Entropy into your application.

## Additional Resources

To learn more about how the protocol works, please see [Protocol

Design](https://docs.pyth.network/entropy/protocol-design).

## Reference Material

- [Protocol Design](https://docs.pyth.network/entropy/protocol-design)

- [Contract

Addresses](https://docs.pyth.network/entropy/contract-addresses)

- [Error Codes](https://docs.pyth.network/entropy/error-codes)

- [Entropy Debugger[(opens in a new

tab)]{.underline}](https://entropy-debugger.pyth.network/) -

Interactive tool for diagnosing callback issues

# Create your first Entropy app on EVM

In this tutorial we will implement and deploy a coin flip contract which

will use entropy to generate a random output.

## Preliminaries

Before we start, please make sure you have the following tools

installed:

Foundry

- [https://book.getfoundry.sh/getting-started/installation[(opens in a

new

tab)]{.underline}](https://book.getfoundry.sh/getting-started/installation).

- Run forge \--version to confirm it is installed. You should get an

output similar to forge 0.2.0 (23aa303

2023-11-28T00:35:15.730515000Z)

Node (version \> 18) - [https://nodejs.org/en/download[(opens in a new

tab)]{.underline}](https://nodejs.org/en/download)

- Run node -v to confirm. You should get an output with version

\>= v18.0.0.

## Getting Started

Create a directory named coin-flip in your filesystem. Open a terminal

with coin-flip as the working directory and run forge init contracts to

create a new Solidity project. You will see a new directory

in coin-flip named contracts. contracts/src is where all your contract

code will be.

Run cd contracts to make it your terminal's working directory --- the

following commands will need to be run from here.

Next, install the Pyth Entropy SDK by running the following commands.

npm init -ynpm install \@pythnetwork/entropy-sdk-solidity

Add a remappings.txt file to contracts directory with the following

content.

\@pythnetwork/entropy-sdk-solidity/=node\_modules/@pythnetwork/entropy-sdk-solidity

## Implementation

Create a new file CoinFlip.sol in contracts/src directory and add the

following code into it to start

// SPDX-License-Identifier: UNLICENSEDpragma solidity \^0.8.13;import

\"@pythnetwork/entropy-sdk-solidity/IEntropy.sol\";import

\"@pythnetwork/entropy-sdk-solidity/IEntropyConsumer.sol\"; contract

CoinFlip is IEntropyConsumer { event FlipRequested(uint64

sequenceNumber); event FlipResult(uint64 sequenceNumber, bool isHeads);

IEntropy entropy; address provider; constructor(address \\_entropy,

address \\_provider) { entropy = IEntropy(\\_entropy); provider =

\\_provider; } // This method is required by the IEntropyConsumer

interface function getEntropy() internal view override returns (address)

{ return address(entropy); }}

The code implements aCoinFlip contract which inherits

the IEntropyConsumer interface. We have also defined some events,

properties and a constructor to instantiate the contract. One of the

properties is of type IEntropy which is an interface imported from the

Entropy SDK. The constructor also takes a provider argument. We will see

how to populate these later.

### Request a coin flip

Copy the following code into CoinFlip.sol.

contract CoinFlip { // \... prior code omitted function request(bytes32

userRandomNumber) external payable { // get the required fee uint128

requestFee = entropy.getFee(provider); // check if the user has sent

enough fees if (msg.value \< requestFee) revert(\"not enough fees\"); //

pay the fees and request a random number from entropy uint64

sequenceNumber = entropy.requestWithCallback{ value: requestFee }(

provider, userRandomNumber ); // emit event emit

FlipRequested(sequenceNumber); }}

Users will invoke the request method to initiate a coin flip with a

request fee and passes in a userRandomNumber argument --- we'll see how

to generate this later. The method first retrieves the fee required to

request a random number from Entropy. It then include the fee in

the requestWithCallback method call to entropy. Finally, the method

emits a FlipRequested event with a sequenceNumber. This event is also

defined in the code snippet above.

### Handle the callback

Copy the following code into CoinFlip.sol.

contract CoinFlip { // \... prior code omitted function entropyCallback(

uint64 sequenceNumber, // If your app uses multiple providers, you can

use this argument // to distinguish which one is calling the app back.

This app only // uses one provider so this argument is not used. address

\\_providerAddress, bytes32 randomNumber ) internal override { bool

isHeads = uint256(randomNumber) % 2 == 0; emit

FlipResult(sequenceNumber, isHeads); }}

Implement entropyCallback method which is required by

the IEntropyConsumer Interface. Entropy calls back this method to

fulfill a request. Entropy will call back this method with

the sequenceNumber of the request, the providerAddress from which the

random number was requested and the generated randomNumber. Finally, the

method emits a FlipResult event with the result of the flip.

Yay! you have successfully implemented a coin flip contract.

## Deploy

First, create a new wallet

cast wallet new

This command will generate a new Ethereum keypair, producing output

similar to the following. Note that the address and private key will be

different hexadecimal values.

Successfully created new keypair.Address:

0xB806824fdA4b2b6631e9B87a86d42C9dfd04D129Private key:

0x0d510c72fd2279155c717eb433ae598a83cfb34b09c2ada86bc424b481082023

We will export the values from the command above as environment

variables to simplify the commands below. We will also export the RPC

URL of the network. Run the following commands in your shell

substituting the address and private key in the indicated places:

export ADDRESS=\<address from above\>export PRIVATE\_KEY=\<your private

key from above\>export RPC\_URL=\"https://sepolia.optimism.io\"

Next, use the Superchain Faucet[(opens in a new tab)]{.underline} to

claim some test Sepolia ETH. Paste the address from the command above

into the faucet to get your ETH. You can verify that the ETH has arrived

in your wallet by running the command

cast balance \$ADDRESS -r \$RPC\_URL -e

The final step before deploying is to get the arguments for the

contract\'s constructor: the [Entropy contract address[(opens in a new

tab)]{.underline}](https://docs.pyth.network/entropy/contract-addresses) for

Optimism Sepolia and [the Provider address[(opens in a new

tab)]{.underline}](https://docs.pyth.network/entropy/contract-addresses).

We will also export these values as environment variables for

convenience:

export ENTROPY\_ADDRESS=0x4821932D0CDd71225A6d914706A621e0389D7061export

PROVIDER\_ADDRESS=0x6CC14824Ea2918f5De5C2f75A9Da968ad4BD6344

Finally, let\'s deploy the contracts. Run the following command:

forge create src/CoinFlip.sol:CoinFlip \\\--private-key \$PRIVATE\_KEY

\\\--rpc-url \$RPC\_URL \\\--constructor-args \$ENTROPY\_ADDRESS

\$PROVIDER\_ADDRESS

You should see an output similar to:

\[⠢\] Compiling\...\[⠔\] Compiling 28 files with 0.8.23\[⠑\] Solc 0.8.23

finished in 3.40sCompiler run successful!Deployer:

0xfa57d0f2CBDA2729273F2a431E4FeDAc656d0402Deployed to:

0x8676ba0Dd492AB9813BC21D5Dce318427d1d73aeTransaction hash:

0x2178aa6d402c94166a93e81822248d00dd003827675ebd49b3c542970f5a0189

Let's export the coin flip contract address as environment variable for

later use:

export COINFLIP\_ADDRESS=\<Deployed to address from above\>

Congratulations you have successfully implemented and deployed a

CoinFlip contract.

## Interact from Javascript

Next, let's interact with the CoinFlip contract from Javascript. Create

a new directory inside coin-flip named app. Run cd app to make it your

terminal's working directory --- the following commands will need to be

run from here.

Run the following to initialise a new project and install required

libraries.

npm init -ynpm install web3 \@pythnetwork/entropy-sdk-solidity

Create a script.js file in app and add the following code to the script.

const { Web3 } = require(\"web3\");const CoinFlipAbi =

require(\"../contracts/out/CoinFlip.sol/CoinFlip.json\");const

EntropyAbi =

require(\"@pythnetwork/entropy-sdk-solidity/abis/IEntropy.json\"); async

function main() { const web3 = new Web3(process.env\[\"RPC\_URL\"\]);

const { address } = web3.eth.accounts.wallet.add(

process.env\[\"PRIVATE\_KEY\"\] )\[0\]; web3.eth.defaultBlock =

\"finalized\"; const coinFlipContract = new web3.eth.Contract(

CoinFlipAbi.abi, process.env\[\"COINFLIP\_ADDRESS\"\] ); const

entropyContract = new web3.eth.Contract( EntropyAbi,

process.env\[\"ENTROPY\_ADDRESS\"\] );} main();

The code above imports the required libraries and defines a main method.

In main we initialize web3 contracts that help us interact with the coin

flip and entropy contracts. At the end, the script calls the main

method.

Next, add the following code to the main method to request a flip from

the CoinFlip contract.

async main() { // \... prior code omitted // Request a random number //

Generate user random number const userRandomNumber =

web3.utils.randomHex(32); const fee = await

entropyContract.methods.getFee(process.env\[\"PROVIDER\_ADDRESS\"\]).call()

console.log(\`fee : \${fee}\`); const requestReceipt = await

coinFlipContract.methods .request(userRandomNumber) .send({ value: fee,

from: address, }); console.log(\`request tx :

\${requestReceipt.transactionHash}\`); // Read the sequence number for

the request from the transaction events. const sequenceNumber =

requestReceipt.events.FlipRequested.returnValues.sequenceNumber;

console.log(\`sequence : \${sequenceNumber}\`); }

The code snippet above generates a random number. The code calls the

Entropy contract to get the fee required for requesting a random number.

Then it calls the request method of the CoinFlip contract with

the userRandomNumber as an argument and the required fee. Finally, the

code reads the sequenceNumber from the FlipRequested event emitted by

the CoinFlip contract.

Finally, add the following code snippet to get the flip result.

async main() { // \... prior code omitted let fromBlock =

requestReceipt.blockNumber; const intervalId = setInterval(async () =\>

{ const currentBlock = await web3.eth.getBlockNumber(); if(fromBlock \>

currentBlock) { return; } // Get \'FlipResult\' events emitted by the

CoinFlip contract for given block range. const events = await

coinFlipContract.getPastEvents(\"FlipResult\", { fromBlock: fromBlock,

toBlock: currentBlock, }); fromBlock = currentBlock + 1n; // Find the

event with the same sequence number as the request. const event =

events.find(event =\> event.returnValues.sequenceNumber ===

sequenceNumber); // If the event is found, log the result and stop

polling. if(event !== undefined) { console.log(\`result :

\${event.returnValues.isHeads ? \'Heads\' : \'Tails\'}\`);

clearInterval(intervalId); } }, 1000);}

The code above polls for new FlipResult events emitted by the CoinFlip

contract. It checks if the event has the same sequenceNumber as the

request. If it does, it logs the result and stops polling.

That's it, Let's run the script with the command node script.js . You

should get an output similar to:

fee : 101request tx :

0xde0dce36a3c149b189aba8b29cad98375a62a811e65efdae28b28524da59cfb6sequence

: 42result : Tails

Note that: the script can fail due to some rpc issues. You can run the

script again to get the expected result.

## Next Steps

Congratulations! You\'ve built your first app using Entropy. In this

tutorial, we created a Solidity contract that generates a random flip

using Entropy. We deployed the contract and interacted with it from

Javascript.

You can learn more about Entropy from the following links:

- [Protocol Design[(opens in a new

tab)]{.underline}](https://docs.pyth.network/entropy/protocol-design)

- [Best Practices](https://docs.pyth.network/entropy/best-practices)

# How to Generate Random Numbers in EVM Contracts Using Pyth Entropy

This guide explains how to integrate Pyth Entropy into EVM Contracts to

generate on-chain random numbers. The intended audience for this guide

is developers of any application that needs on-chain randomness, such as

NFT mints or games.

## Install the SDK

Pyth Entropy has a [Solidity SDK[(opens in a new

tab)]{.underline}](https://github.com/pyth-network/pyth-crosschain/tree/main/target\_chains/ethereum/entropy\_sdk/solidity) that

lets your contract interact with the Entropy contract. Install the SDK

using your package manager:

hardhatfoundry

npm install \@pythnetwork/entropy-sdk-solidity

## Setup

The Solidity SDK exports two interfaces:

- [[IEntropyConsumer(opens in a new

tab)]{.underline}](https://github.com/pyth-network/pyth-crosschain/blob/main/target\_chains/ethereum/entropy\_sdk/solidity/IEntropyConsumer.sol) -

The interface that your contract should implement. It makes sure

that your contract is compliant with the Entropy contract.

- [[IEntropy(opens in a new

tab)]{.underline}](https://github.com/pyth-network/pyth-crosschain/blob/main/target\_chains/ethereum/entropy\_sdk/solidity/IEntropy.sol) -

The interface to interact with the Entropy contract. You will need

the address of an Entropy contract on your blockchain. Consult the

current [Entropy contract

addresses](https://docs.pyth.network/entropy/contract-addresses) to

find the address on your chain. Once you have a contract address,

instantiate an IEntropy contract in your solidity contract:

import { IEntropyConsumer } from

\"@pythnetwork/entropy-sdk-solidity/IEntropyConsumer.sol\";import {

IEntropy } from \"@pythnetwork/entropy-sdk-solidity/IEntropy.sol\"; //

\@param entropyAddress The address of the entropy contract.contract

YourContract is IEntropyConsumer { IEntropy public entropy;

constructor(address entropyAddress) { entropy =

IEntropy(entropyAddress); }}

Entropy also requires selecting a \*\*randomness provider\*\*. The

randomness provider is a third-party who participates in the generation

process. Each provider is identified by an address and hosts a keeper

service for fullfilling requests.

The simplest way to choose a provider is to use the [default

provider](https://docs.pyth.network/entropy/contract-addresses). The

default provider for each contract and their corresponding URI is also

listed in the [Entropy contract

addresses](https://docs.pyth.network/entropy/contract-addresses).

You can also get the default provider\'s address by calling

the [[getDefaultProvider(opens in a new

tab)]{.underline}](https://github.com/pyth-network/pyth-crosschain/blob/f8ebeb6af31d98f94ce73edade6da2ebab7b2456/target\_chains/ethereum/entropy\_sdk/solidity/IEntropy.sol#L94) method:

address provider = entropy.getDefaultProvider();

## Usage

To generate a random number, follow these steps.

### 1. Generate a random number

Generate a 32-byte random number on the client side.

web3.jsethers.js

const userRandomNumber = web3.utils.randomHex(32);

### 2. Request a number from Entropy

Invoke the [[requestWithCallback(opens in a new

tab)]{.underline}](https://github.com/pyth-network/pyth-crosschain/blob/main/target\_chains/ethereum/entropy\_sdk/solidity/IEntropy.sol#L83) method

of the IEntropy contract. The requestWithCallback method requires paying

a fee in native gas tokens which is configured per-provider.

The fees differs for every chain and can be found at the [Current

Fees](https://docs.pyth.network/entropy/current-fees) page.\

You can use the onchain method [[getFee(opens in a new

tab)]{.underline}](https://github.com/pyth-network/pyth-crosschain/blob/main/target\_chains/ethereum/entropy\_sdk/solidity/IEntropy.sol#L101) to

calculate the fee for the default provider and send it as the value of

the requestWithCallback call:

function requestRandomNumber(bytes32 userRandomNumber) external payable

{ uint256 fee = entropy.getFee(entropyProvider); uint64 sequenceNumber =

entropy.requestWithCallback{ value: fee }( entropyProvider,

userRandomNumber );}

This method returns a sequence number and emits

a [[RequestedWithCallback(opens in a new

tab)]{.underline}](https://github.com/pyth-network/pyth-crosschain/blob/main/target\_chains/ethereum/entropy\_sdk/solidity/EntropyEvents.sol#L10) event.

You can store this sequence number to identify the request in next step.

### 3. Implement callback for Entropy

pragma solidity \^0.8.0; import { IEntropyConsumer } from

\"@pythnetwork/entropy-sdk-solidity/IEntropyConsumer.sol\";import {

IEntropy } from \"@pythnetwork/entropy-sdk-solidity/IEntropy.sol\";

contract YourContract is IEntropyConsumer { IEntropy entropy; // \@param

entropyAddress The address of the entropy contract. constructor(address

entropyAddress) { entropy = IEntropy(entropyAddress); } // \@param

userRandomNumber The random number generated by the user. function

requestRandomNumber(bytes32 userRandomNumber) external payable { // Get

the default provider and the fee for the request address entropyProvider

= entropy.getDefaultProvider(); uint256 fee =

entropy.getFee(entropyProvider); // Request the random number with the

callback uint64 sequenceNumber = entropy.requestWithCallback{ value: fee

}( entropyProvider, userRandomNumber ); // Store the sequence number to

identify the callback request } // \@param sequenceNumber The sequence

number of the request. // \@param provider The address of the provider

that generated the random number. If your app uses multiple providers,

you can use this argument to distinguish which one is calling the app

back. // \@param randomNumber The generated random number. // This

method is called by the entropy contract when a random number is

generated. // This method \\*\\*must\\*\\* be implemented on the same

contract that requested the random number. // This method should

\\*\\*never\\*\\* return an error \-- if it returns an error, then the

keeper will not be able to invoke the callback. // If you are having

problems receiving the callback, the most likely cause is that the

callback is erroring. // See the callback debugging guide here to

identify the error

https://docs.pyth.network/entropy/debug-callback-failures function

entropyCallback( uint64 sequenceNumber, address provider, bytes32

randomNumber ) internal override { // Implement your callback logic

here. } // This method is required by the IEntropyConsumer interface. //

It returns the address of the entropy contract which will call the

callback. function getEntropy() internal view override returns (address)

{ return address(entropy); }}

When the final random number is ready to use, the entropyCallback

function will be called by the Entropy contract. This will happen in a

separate transaction submitted by the requested provider.

⚠️

The entropyCallback function should \*\*never\*\* return an error. If it

returns an error, the keeper will not be able to invoke the callback. If

you are having problems receiving the callback, please see [Debugging

Callback

Failures](https://docs.pyth.network/entropy/debug-callback-failures).

## Additional Resources

You may find these additional resources helpful while integrating Pyth

Entropy into your EVM contract.

### Debug Callback Failures

Check how to [Debug Callback

Failures](https://docs.pyth.network/entropy/debug-callback-failures) if

you are having trouble getting the callback to run.

### Pyth Entropy Contract Addresses

Consult the [Entropy contract

addresses](https://docs.pyth.network/entropy/contract-addresses) to find

the Entropy contract address on your chain.

### Current Fees

Check the [Current

Fees](https://docs.pyth.network/entropy/current-fees) to find the

current fee for each provider on your chain.

### Best Practices

Check out the [Best

Practices](https://docs.pyth.network/entropy/best-practices) guide for

tips to limit gas usage, or generate multiple random numbers in a single

transaction.

# Debug Callback Failures

🔍

\*\*Quick Debug Tool\*\*: Use the [Entropy Debugger[(opens in a new

tab)]{.underline}](https://entropy-debugger.pyth.network/) to quickly

diagnose and resolve callback issues.

This guide explains how to identify and resolve issues with the Entropy

callback. The intended audience for this guide is developers who have

made an Entropy random number request, but their application hasn\'t

received a callback.

## Dependencies

This guide uses [Foundry[(opens in a new

tab)]{.underline}](https://book.getfoundry.sh/getting-started/installation) to

submit transactions to the blockchain. Please install Foundry before

continuing.

## Run the Callback

Developers can run the Entropy callback themselves to see the reason for

the failure. To run the callback, invoke the revealWithCallback function

on the Entropy contract on your blockchain. The function has the

following signature:

function revealWithCallback(address provider, uint64 sequenceNumber,

bytes32 userRandomNumber, bytes32 providerRevelation)

This call requires the chain ID, contract address, and four arguments.

The chain ID and contract address can be retrieved from [Contract

Addresses](https://docs.pyth.network/entropy/contract-addresses). Export

these values as environment variables for later use:

export CHAIN\_ID=blastexport

ENTROPY\_ADDRESS=0x5744Cbf430D99456a0A8771208b674F27f8EF0Fb

Three of the arguments can be retrieved from the request transaction\'s

event logs. Look at the event logs of the request transaction in a block

explorer. You should see a RequestedWithCallback event emitted from the

Entropy contract, similar to the one below:

![Example Entropy event logs](media/image1.png){width="6.5in"

height="3.0319444444444446in"}

Copy the following values from the event into environment variables:

export PROVIDER=0x52DeaA1c84233F7bb8C8A45baeDE41091c616506export

SEQUENCE\_NUMBER=264793# First row of the \"Data\" field with a 0x

prefixexport

USER\_RANDOM\_NUMBER=0xb1db70cfe95d9e823ac591362921c20eb45f136dbf417362f0a045dc7100519e

Next, collect the provider\'s revelation. Every provider exposes a URL

that allows users to get the provider\'s revelation for unfulfilled

requests. The URLs for the default providers are as follows:

-----------------------------------------------------------------------

\*\*Network \*\*Provider URL\*\*

type\*\*

--------------- -------------------------------------------------------

testnet https://fortuna-staging.dourolabs.app/

mainnet https://fortuna.dourolabs.app

-----------------------------------------------------------------------

Retrieve

the /v1/chains/\$CHAIN\_ID/revelations/\$SEQUENCE\_NUMBER endpoint from

this server:

curl

https://fortuna.dourolabs.app/v1/chains/\$CHAIN\_ID/revelations/\$SEQUENCE\_NUMBER

⚠️

The chain ID is the string name of the chain and not the EVM chain ID.

The chain ids are available [here[(opens in a new

tab)]{.underline}](https://fortuna.dourolabs.app/docs/#/crate%3A%3Aapi/chain\_ids).

This endpoint will return the provider\'s revelation as a hexadecimal

value, such as:

{ \"value\": { \"encoding\": \"hex\", \"data\":

\"5d4bfa3abeaf15fe8b7771c74c0e3e210096015632831460870bc5374e05d4d8\" }}

Export this value as an environment variable (prefixed with 0x):

export

PROVIDER\_REVELATION=0x5d4bfa3abeaf15fe8b7771c74c0e3e210096015632831460870bc5374e05d4d8

Finally, submit the transaction to invoke revealWithCallback:

cast send \$ENTROPY\_ADDRESS \'revealWithCallback(address, uint64,

bytes32, bytes32)\' \$PROVIDER \$SEQUENCE\_NUMBER \$USER\_RANDOM\_NUMBER

\$PROVIDER\_REVELATION

You may also need to provide the -r with an RPC for your chain, plus an

additional argument (such as \--private-key) to specify the wallet to

use to sign the transaction.

## Interpreting the Results

If the transaction succeeds, check how much gas the transaction used. If

the gas exceeds the callback gas limit for your chain as shown

on [Contract

Addresses](https://docs.pyth.network/entropy/contract-addresses), please

reduce the gas usage to allow the provider to invoke the callback in the

future.

If the transaction fails, it typically indicates a failure originating

from the implementation of the callback function. Check your

implementation of the entropyCallback function to ensure that it does

not throw an error when invoked.

# Best Practices

## Limit gas usage on the callback

Keeping the callback function simple is crucial because the entropy

providers limit gas usage. This ensures gas usage is predictable and

consistent, avoiding potential issues with the callback.

For example, if you want to use entropy to generate a random number for

each player in a round of game, you need to make sure that the callback

function works for the maximum number of players that can be in each

round. Otherwise, the callbacks will work for some rounds with fewer

players, but will fail for rounds with more players.

Multiple solutions are possible to address this problem. You can store

the random number received from the callback and either ask users to

submit more transactions after the callback to continue the flow or run

a background crank service to submit the necessary transactions.

The gas limit for each chain is listed on the [contract

addresses](https://docs.pyth.network/entropy/contract-addresses) page.

## Handling callback failures

While the default entropy provider is highly reliable, in rare cases a

callback might not be received. This typically happens when there\'s an

issue with your contract\'s callback implementation rather than with the

provider itself. The most common causes are:

1. The callback function is using more gas than the allowed limit

2. The callback function contains logic that throws an error

If you\'re not receiving a callback, you can manually invoke it to

identify the specific issue. This allows you to:

- See if the transaction fails and why

- Check the gas usage against the chain\'s callback gas limit

- Debug your callback implementation

For detailed instructions on how to manually invoke and debug callbacks,

refer to the [Debug Callback

Failures](https://docs.pyth.network/entropy/debug-callback-failures) guide.

## Generating random values within a specific range

You can map the random number provided by Entropy into a smaller range

using the solidity [modulo operator[(opens in a new

tab)]{.underline}](https://docs.soliditylang.org/en/latest/types.html#modulo).

Here is a simple example of how to map a random number provided by

Entropy into a range between minRange and maxRange (inclusive).

// Maps a random number into a range between minRange and maxRange

(inclusive)function mapRandomNumber( bytes32 randomNumber, int256

minRange, int256 maxRange) internal returns (int256) { uint256 range =

uint256(maxRange - minRange + 1); return minRange +

int256(uint256(randomNumber) % range);}

Notice that using the modulo operator can distort the distribution of

random numbers if it\'s not a power of 2. This is negligible for small

and medium ranges, but it can be noticeable for large ranges. For

example, if you want to generate a random number between 1 and 52, the

probability of having value 5 is approximately 10\^-77 higher than the

probability of having value 50 which is infinitesimal.

## Generating multiple random values in a single transaction

If you need to generate multiple random values in a single transaction,

you can hash the random input provided by Entropy with a unique

identifier for each random number.

In the following example, mapRandomNumber is used to generate 6 random

attributes for a character.

function generateAttributes(bytes32 randomNumber) internal { int256

strength = mapRandomNumber( keccak256(abi.encodePacked(randomNumber,

\"strength\")), 15, 20 ); int256 stamina = mapRandomNumber(

keccak256(abi.encodePacked(randomNumber, \"stamina\")), 10, 15 ); int256

agility = mapRandomNumber( keccak256(abi.encodePacked(randomNumber,

\"agility\")), 5, 15 ); int256 stealth = mapRandomNumber(

keccak256(abi.encodePacked(randomNumber, \"stealth\")), 0, 5 ); int256

positionX = mapRandomNumber( keccak256(abi.encodePacked(randomNumber,

\"positionX\")), -100, 100 ); int256 positionY = mapRandomNumber(

keccak256(abi.encodePacked(randomNumber, \"positionY\")), -100, 100 );}

# Entropy Contract Addresses on EVM

## Mainnets

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

\*\*Chain \*\*Entropy Contract Address\*\* \*\*Reveal \*\*Gas

Id\*\* Delay\*\* Limit\*\*

----------- --------------------------------------------------------------------------------------------------------------------------------------------------------- ----------- ---------

abstract [[0x5a4a369F4db5df2054994AF031b7b23949b98c0e]{.underline}](https://abscan.org/address/0x5a4a369F4db5df2054994AF031b7b23949b98c0e) 0 block 500K

apechain [[0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320]{.underline}](https://apechain.calderaexplorer.xyz/address/0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320) 1 block 500K

arbitrum [[0x7698E925FfC29655576D0b361D75Af579e20AdAc]{.underline}](https://arbiscan.io/address/0x7698E925FfC29655576D0b361D75Af579e20AdAc) 6 blocks 2.5M

b3 [[0x5744Cbf430D99456a0A8771208b674F27f8EF0Fb]{.underline}](https://explorer.b3.fun/address/0x5744Cbf430D99456a0A8771208b674F27f8EF0Fb) 1 block 500K

base [[0x6E7D74FA7d5c90FEF9F0512987605a6d546181Bb]{.underline}](https://basescan.org/address/0x6E7D74FA7d5c90FEF9F0512987605a6d546181Bb) 1 block 500K

berachain [[0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320]{.underline}](https://berascan.com/address/0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320) 1 block 2.5M

blast [[0x5744Cbf430D99456a0A8771208b674F27f8EF0Fb]{.underline}](https://blastscan.io/address/0x5744Cbf430D99456a0A8771208b674F27f8EF0Fb) 1 block 500K

etherlink [[0x23f0e8FAeE7bbb405E7A7C3d60138FCfd43d7509]{.underline}](https://explorer.etherlink.com/address/0x23f0e8FAeE7bbb405E7A7C3d60138FCfd43d7509) 1 block 15M

hyperevm [[0xfA25E653b44586dBbe27eE9d252192F0e4956683]{.underline}](https://hyperliquid.cloud.blockscout.com/address/0xfA25E653b44586dBbe27eE9d252192F0e4956683) 0 block 500K

kaia [[0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320]{.underline}](https://kaiascan.io/address/0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320) 1 block 500K

optimism [[0xdF21D137Aadc95588205586636710ca2890538d5]{.underline}](https://optimistic.etherscan.io/address/0xdF21D137Aadc95588205586636710ca2890538d5) 2 blocks 500K

sanko [[0x5744Cbf430D99456a0A8771208b674F27f8EF0Fb]{.underline}](https://explorer.sanko.xyz/address/0x5744Cbf430D99456a0A8771208b674F27f8EF0Fb) 1 block 500K

sei-evm [[0x98046Bd286715D3B0BC227Dd7a956b83D8978603]{.underline}](https://seitrace.com/address/0x98046Bd286715D3B0BC227Dd7a956b83D8978603?chain=pacific-1) 1 block 500K

soneium [[0x0708325268dF9F66270F1401206434524814508b]{.underline}](https://soneium.blockscout.com/address/0x0708325268dF9F66270F1401206434524814508b) 1 block 500K

sonic [[0x36825bf3fbdf5a29e2d5148bfe7dcf7b5639e320]{.underline}](https://sonicscan.org/address/0x36825bf3fbdf5a29e2d5148bfe7dcf7b5639e320) 1 block 500K

story [[0xdF21D137Aadc95588205586636710ca2890538d5]{.underline}](https://storyscan.xyz/address/0xdF21D137Aadc95588205586636710ca2890538d5) 0 block 500K

unichain [[0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320]{.underline}](https://unichain.blockscout.com/address/0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320) 1 block 500K

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

\*\*The default provider for above mainnet chains

is 0x52DeaA1c84233F7bb8C8A45baeDE41091c616506.\*\*

The default provider on mainnet has a reveal delay to avoid changes on

the outcome of the Entropy request because of block reorgs. The reveal

delay shows how many blocks should be produced after the block including

the request transaction in order to reveal and submit a callback

transaction.

The default provider fulfills the request by sending a transaction with

a gas limit as mentioned in above table. Entropy callbacks the consumer

as part of this transaction.

## Testnets

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

\*\*Chain Id\*\* \*\*Entropy Contract Address\*\* \*\*Reveal \*\*Gas

Delay\*\* Limit\*\*

------------------------ --------------------------------------------------------------------------------------------------------------------------------------------------------------- ---------- ---------

abstract-testnet [[0x858687fD592112f7046E394A3Bf10D0C11fF9e63]{.underline}](https://explorer.testnet.abs.xyz/address/0x858687fD592112f7046E394A3Bf10D0C11fF9e63) 0 block 500K

apechain-testnet [[0x23f0e8FAeE7bbb405E7A7C3d60138FCfd43d7509]{.underline}](https://curtis.explorer.caldera.xyz/address/0x23f0e8FAeE7bbb405E7A7C3d60138FCfd43d7509) 1 block 500K

arbitrum-sepolia [[0x549Ebba8036Ab746611B4fFA1423eb0A4Df61440]{.underline}](https://sepolia.arbiscan.io/address/0x549Ebba8036Ab746611B4fFA1423eb0A4Df61440) 6 blocks 2.5M

b3-testnet [[0x5744Cbf430D99456a0A8771208b674F27f8EF0Fb]{.underline}](https://sepolia.explorer.b3.fun/address/0x5744Cbf430D99456a0A8771208b674F27f8EF0Fb) 1 block 500K

base-sepolia [[0x41c9e39574F40Ad34c79f1C99B66A45eFB830d4c]{.underline}](https://base-sepolia.blockscout.com/address/0x41c9e39574F40Ad34c79f1C99B66A45eFB830d4c) 1 block 500K

berachain-bepolia [[0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320]{.underline}](https://bepolia.beratrail.io/address/0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320) 1 block 500K

blast-testnet [[0x98046Bd286715D3B0BC227Dd7a956b83D8978603]{.underline}](https://testnet.blastscan.io/address/0x98046Bd286715D3B0BC227Dd7a956b83D8978603) 1 block 500K

etherlink-testnet [[0x23f0e8FAeE7bbb405E7A7C3d60138FCfd43d7509]{.underline}](https://testnet.explorer.etherlink.com/address/0x23f0e8FAeE7bbb405E7A7C3d60138FCfd43d7509) 1 block 15M

kaia-testnet [[0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320]{.underline}](https://kairos.kaiascan.io/address/0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320) 1 block 500K

monad-testnet [[0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320]{.underline}](https://testnet.monadexplorer.com/address/0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320) 2 blocks 500K

optimism-sepolia [[0x4821932D0CDd71225A6d914706A621e0389D7061]{.underline}](https://optimism-sepolia.blockscout.com/address/0x4821932D0CDd71225A6d914706A621e0389D7061) 2 blocks 500K

sanko-testnet [[0x5744Cbf430D99456a0A8771208b674F27f8EF0Fb]{.underline}](https://sanko-arb-sepolia.explorer.caldera.xyz/address/0x5744Cbf430D99456a0A8771208b674F27f8EF0Fb) 1 block 500K

sei-evm-testnet [[0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320]{.underline}](https://seitrace.com/address/0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320?chain=atlantic-2) 1 block 500K

soneium-minato-testnet [[0x23f0e8FAeE7bbb405E7A7C3d60138FCfd43d7509]{.underline}](https://explorer-testnet.soneium.org/address/0x23f0e8FAeE7bbb405E7A7C3d60138FCfd43d7509) 1 block 500K

sonic-blaze-testnet [[0xebe57e8045f2f230872523bbff7374986e45c486]{.underline}](https://blaze.soniclabs.com/address/0xebe57e8045f2f230872523bbff7374986e45c486) 1 block 500K

story-testnet [[0x5744Cbf430D99456a0A8771208b674F27f8EF0Fb]{.underline}](https://aeneid.storyscan.xyz/address/0x5744Cbf430D99456a0A8771208b674F27f8EF0Fb) 0 block 500K

tabi-testnet [[0xEbe57e8045F2F230872523bbff7374986E45C486]{.underline}](https://testnetv2.tabiscan.com/address/0xEbe57e8045F2F230872523bbff7374986E45C486) 0 block 500K

unichain-sepolia [[0x8D254a21b3C86D32F7179855531CE99164721933]{.underline}](https://unichain-sepolia.blockscout.com/address/0x8D254a21b3C86D32F7179855531CE99164721933) 1 block 500K

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

\*\*The default provider for above testnet chains

is 0x6CC14824Ea2918f5De5C2f75A9Da968ad4BD6344.\*\*

The default provider on testnet has reveal delays identical to the

corresponding mainnet chains to ensure consistent behavior across

environments.

The default provider fulfills the request by sending a transaction with

a gas limit as mentioned in above table. Entropy callbacks the consumer

as part of this transaction.

The following tables shows the total fees payable when using

the \*\*default provider\*\*. Note that the fees shown below will vary over

time with prevailing gas prices on each chain.

## Mainnet

⚠️

The fees for mainnet are dynamically set. Always use the onchain

method entropy.getFee(entropyProvider) to get the current fee.

---------------------------------------------------------------------------------------------------------------------------------------------------------------

\*\*Chain Id\*\* \*\*Fee\*\*

--------------------------------------------------------------------------------------------------------- -----------------------------------------------------

[abstract](https://abscan.org/address/0x5a4a369F4db5df2054994AF031b7b23949b98c0e) 0.000019006155000001 \*\*ETH\*\*

[apechain](https://apechain.calderaexplorer.xyz/address/0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320) 0.020000000000000001 \*\*APE\*\*

[arbitrum](https://arbiscan.io/address/0x7698E925FfC29655576D0b361D75Af579e20AdAc) 0.000044476200000001 \*\*ETH\*\*

[b3](https://explorer.b3.fun/address/0x5744Cbf430D99456a0A8771208b674F27f8EF0Fb) 0.000005000000000001 \*\*ETH\*\*

[base](https://basescan.org/address/0x6E7D74FA7d5c90FEF9F0512987605a6d546181Bb) 0.000005000000000001 \*\*ETH\*\*

[berachain](https://berascan.com/address/0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320) 0.001250000000000001 \*\*BERA\*\*

[blast](https://blastscan.io/address/0x5744Cbf430D99456a0A8771208b674F27f8EF0Fb) 0.000005000000000001 \*\*ETH\*\*

[etherlink](https://explorer.etherlink.com/address/0x23f0e8FAeE7bbb405E7A7C3d60138FCfd43d7509) 0.018000000000000001 \*\*XTZ\*\*

[hyperevm](https://hyperliquid.cloud.blockscout.com/address/0xfA25E653b44586dBbe27eE9d252192F0e4956683) 0.002988000000000001 \*\*HYPE\*\*

[kaia](https://kaiascan.io/address/0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320) 0.100000000000000001 \*\*KLAY\*\*

[optimism](https://optimistic.etherscan.io/address/0xdF21D137Aadc95588205586636710ca2890538d5) 0.000005000000000001 \*\*ETH\*\*

[sanko](https://explorer.sanko.xyz/address/0x5744Cbf430D99456a0A8771208b674F27f8EF0Fb) 0.001000000000000001 \*\*DMT\*\*

[sei-evm](https://seitrace.com/address/0x98046Bd286715D3B0BC227Dd7a956b83D8978603?chain=pacific-1) 0.050000000000000001 \*\*SEI\*\*

[soneium](https://soneium.blockscout.com/address/0x0708325268dF9F66270F1401206434524814508b) 0.000005000000000001 \*\*ETH\*\*

[sonic](https://sonicscan.org/address/0x36825bf3fbdf5a29e2d5148bfe7dcf7b5639e320) 0.036000001800000001 \*\*S\*\*

[story](https://storyscan.xyz/address/0xdF21D137Aadc95588205586636710ca2890538d5) 0.002000000000000001 \*\*IP\*\*

[unichain](https://unichain.blockscout.com/address/0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320) 0.000005000000000001 \*\*ETH\*\*

---------------------------------------------------------------------------------------------------------------------------------------------------------------

## Testnet

ℹ️

The fees for testnets kept deliberately low and different from the

mainnet fees.

-------------------------------------------------------------------------------------------------------------------------------------------------------------

\*\*Chain Id\*\* \*\*Fee\*\*

-------------------------------------------------------------------------------------------------------------------- ----------------------------------------

[abstract-testnet](https://explorer.testnet.abs.xyz/address/0x858687fD592112f7046E394A3Bf10D0C11fF9e63) 0.000010501155000001 \*\*ETH\*\*

[apechain-testnet](https://curtis.explorer.caldera.xyz/address/0x23f0e8FAeE7bbb405E7A7C3d60138FCfd43d7509) 0.000015000000000001 \*\*APE\*\*

[arbitrum-sepolia](https://sepolia.arbiscan.io/address/0x549Ebba8036Ab746611B4fFA1423eb0A4Df61440) 0.000372357000000001 \*\*ETH\*\*

[b3-testnet](https://sepolia.explorer.b3.fun/address/0x5744Cbf430D99456a0A8771208b674F27f8EF0Fb) 0.000015000000000001 \*\*ETH\*\*

[base-sepolia](https://base-sepolia.blockscout.com/address/0x41c9e39574F40Ad34c79f1C99B66A45eFB830d4c) 0.000015000000000001 \*\*ETH\*\*

[berachain-bepolia](https://bepolia.beratrail.io/address/0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320) 0.000015000000000001 \*\*BERA\*\*

[blast-testnet](https://testnet.blastscan.io/address/0x98046Bd286715D3B0BC227Dd7a956b83D8978603) Loading\...

[etherlink-testnet](https://testnet.explorer.etherlink.com/address/0x23f0e8FAeE7bbb405E7A7C3d60138FCfd43d7509) 0.062269294500000001 \*\*XTZ\*\*

[kaia-testnet](https://kairos.kaiascan.io/address/0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320) 0.240000000000000001 \*\*KLAY\*\*

[monad-testnet](https://testnet.monadexplorer.com/address/0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320) 0.027000001350000001 \*\*MON\*\*

[optimism-sepolia](https://optimism-sepolia.blockscout.com/address/0x4821932D0CDd71225A6d914706A621e0389D7061) 0.000015000000000001 \*\*ETH\*\*

[sanko-testnet](https://sanko-arb-sepolia.explorer.caldera.xyz/address/0x5744Cbf430D99456a0A8771208b674F27f8EF0Fb) 0.000360001800000001 \*\*DMT\*\*

[sei-evm-testnet](https://seitrace.com/address/0x36825bf3Fbdf5a29E2d5148bfe7Dcf7B5639e320?chain=atlantic-2) 0.010000000000000001 \*\*SEI\*\*

[soneium-minato-testnet](https://explorer-testnet.soneium.org/address/0x23f0e8FAeE7bbb405E7A7C3d60138FCfd43d7509) 0.000015000000000001 \*\*ETH\*\*

[sonic-blaze-testnet](https://blaze.soniclabs.com/address/0xebe57e8045f2f230872523bbff7374986e45c486) 0.000720001800000001 \*\*S\*\*

[story-testnet](https://aeneid.storyscan.xyz/address/0x5744Cbf430D99456a0A8771208b674F27f8EF0Fb) 0.000900000004200001 \*\*IP\*\*

[tabi-testnet](https://testnetv2.tabiscan.com/address/0xEbe57e8045F2F230872523bbff7374986E45C486) 0.011040000000000001 \*\*TABI\*\*

[unichain-sepolia](https://unichain-sepolia.blockscout.com/address/0x8D254a21b3C86D32F7179855531CE99164721933) 0.000059963618400001 \*\*ETH\*\*

-------------------------------------------------------------------------------------------------------------------------------------------------------------

# Error Codes

The following table contains the errors used in the Pyth Network\'s

Entropy [EVM contracts[(opens in a new

tab)]{.underline}](https://github.com/pyth-network/pyth-crosschain/blob/d290f4ec47a73636cf77711f5f68c3455bb8a8ca/target\_chains/ethereum/contracts/contracts/entropy/Entropy.sol).

This information is derived from EntropyErrors.sol[(opens in a new

tab)]{.underline} in the Pyth EntropySDK and can be used to decode error

codes programmatically.

-------------------------------------------------------------------------------------------

\*\*Error \*\*Error\*\* \*\*Error Description\*\*

Codes\*\*

------------ ----------------------------- ------------------------------------------------

0xd82dd966 AssertionFailure() Contract invariant failed.

0xda041bdf ProviderAlreadyRegistered() Provider already registered.

0xdf51c431 NoSuchProvider() Requested Provider does not exist.

0xc4237352 NoSuchRequest() Request does not exist or the request has been

fulfilled.

0x3e515085 OutOfRandomness() Provider is out of committed random numbers.

0x025dbdd4 InsufficientFee() Request fee is insufficient.

0xb8be1a8d IncorrectRevelation() Revelation does not match commitment.

0xb463ce7a InvalidUpgradeMagic() Governance message is invalid.

0x82b42900 Unauthorized() msg.sender is not allowed to invoke this method.

0x92555c0e BlockhashUnavailable() Blockhash is unavailable.

0x50f0dc92 InvalidRevealCall() Invalid reveal call method. If a request was

made using requestWithCallback, request should

be fulfilled using revealWithCallbackelse if a

request was made using request, request should

be fulfilled using reveal

0xb28d9c76 LastRevealedTooOld() Last random number revealed is too old.

0x5e5b3f1b UpdateTooOld() More recent commitment already revealed

on-chain.

-------------------------------------------------------------------------------------------

# Example Applications

The [Coin Flip[(opens in a new

tab)]{.underline}](https://github.com/pyth-network/pyth-examples/tree/main/entropy/coin\_flip) example

demonstrates how to build a smart contract that interacts with Pyth

Entropy as well as a typescript client for that application.

# <https://github.com/pyth-network/pyth-examples/tree/main/entropy/coin\_flip> Protocol Design

The Entropy protocol is an extension of a classical commit/reveal

protocol. The original version has the following steps:

1. Two parties A and B each draw secret random

numbers, xA\*xA\*​ and xB\*xB\*​.

2. A and B hash their random numbers and share the

hashes, hA=hash(xA)\*hA\*​=hash(\*xA\*​) and hB=hash(xB)\*hB\*​=hash(\*xB\*​)

3. A and B reveal xA\*xA\*​ and xB\*xB\*​

4. Both parties verify

that hash(xA)=hAhash(\*xA\*​)=\*hA\*​ and hash(xB)=hBhash(\*xB\*​)=\*hB\*​

5. The random number r=hash(xA,xB)\*r\*=hash(\*xA\*​,\*xB\*​)

This protocol has the property that the result is random as long as

either A or B are honest. Thus, neither party needs to trust the other

\-- as long as they are themselves honest, they can ensure that the

result r\*r\* is random.

The diagram below shows the protocol flow:![Entropy Protocol

Flow](media/image2.png){width="6.5in" height="3.654861111111111in"}

Entropy implements a version of this protocol that is optimized for

on-chain usage. The key difference is that one of the participants (the

provider) commits to a sequence of random numbers up-front using a hash

chain. Users of the protocol then simply grab the next random number in

the sequence.

\*\*Setup\*\*: The provider P computes a sequence of N\*N\* random

numbers, xi\*xi\*​ for 0≤i≤N−10≤\*i\*≤\*N\*−1:

- xN−1=random()\*xN\*−1​=random()

- xi=hash(xi+1)\*xi\*​=hash(\*xi\*+1​)

The provider commits to x0\*x\*0​ by posting it to the Entropy contract.

Each random number in the sequence can then be verified against the

previous one in the sequence by hashing it,

i.e., hash(xi)=xi−1hash(\*xi\*​)=\*xi\*−1​

Pyth Entropy uses automatic callbacks to simplify the flow:

- \*\*Request\*\*: To produce a random number, the following steps occur.

1. The user U draws a random number xU\*xU\*​, and submits it to the

contract. The contract generates the

hash hU=hash(xU)\*hU\*​=hash(\*xU\*​) and records both xU\*xU\*​ and hU\*hU\*​.

The contract uses [[constructUserCommitment(opens in a new

tab)]{.underline}](https://github.com/pyth-network/pyth-crosschain/blob/7bccde484f01c19844b7105d63df207a24018957/target\_chains/ethereum/contracts/contracts/entropy/Entropy.sol#L628-L632) to

generate the user\'s commitment.

2. The contract [remembers [hU\*hU\*​]{.underline} and assigns it an

incrementing [\*\*sequence number i\*i\*\*\*(opens in a new

tab)]{.underline}](https://github.com/pyth-network/pyth-crosschain/blob/7bccde484f01c19844b7105d63df207a24018957/target\_chains/ethereum/contracts/contracts/entropy/Entropy.sol#L232-L246),

representing which of the provider\'s random numbers the user will

receive. xU\*xU\*​ is recorded in the [event logs[(opens in a new

tab)]{.underline}](https://github.com/pyth-network/pyth-crosschain/blob/7bccde484f01c19844b7105d63df207a24018957/target\_chains/ethereum/contracts/contracts/entropy/Entropy.sol#L300-L306).

3. After sufficient block confirmations, the provider submits a reveal

transaction with xi\*xi\*​ and xU\*xU\*​ to the contract.

4. The contract

verifies hash(xU)=hUhash(\*xU\*​)=\*hU\*​ and hash(xi)=xi−1hash(\*xi\*​)=\*xi\*−1​ to

prove that xi\*xi\*​ is the i\*i\*\'th random number.

5. If both of the above conditions are satisfied, the random

number r=hash(xi,xU)\*r\*=hash(\*xi\*​,\*xU\*​) is generated and a callback

is made to the requesting contract.

In this flow, providers can refuse revealing xi\*xi\*​ if the final random

number r\*r\* is not in their favor, or they may be able to

access xU\*xU\*​ before on-chain submission (e.g. via mempool) and rotate

their commitments to influence the random number r\*r\*. Of course, both

of these behaviors are detectable and protocols can blacklist providers

that exhibit them.

This protocol has the same security properties as the 2-party randomness

protocol above: as long as either the provider or user is honest, the

number r\*r\* is random.

The diagram below shows the user\'s interaction with the entropy

contract to generate a random number:![Entropy Contract

Flow](media/image3.png){width="6.5in" height="3.654861111111111in"}

Note that providers need to be careful to ensure their off-chain service

isn\'t compromised to reveal the random numbers \-- if this occurs, then

users will be able to influence the random number r\*r\*.

The code of default deployed provider can be found [here[(opens in a new

tab)]{.underline}](https://github.com/pyth-network/pyth-crosschain/tree/7bccde484f01c19844b7105d63df207a24018957/apps/fortuna).

# <https://github.com/pyth-network/pyth-crosschain/tree/7bccde484f01c19844b7105d63df207a24018957/apps/fortuna> Fees

The Entropy protocol has been designed to charge fees on a per-request

basis. The total fee is the sum of the provider fee, which is determined

by the individual provider on a per-blockchain basis, and the protocol

fee, which is subject to Pyth governance decisions also on a per-chain

basis. These are accessible via the smart contract functions.

Providers can withdraw their fees from the contract whenever they want.

The allocation of the fees collected by the Entropy protocol is governed

by the collective decisions of its governance body. All fees are

denominated in the native token of the respective blockchain, ensuring a

seamless and integrated operation.

Note that protocols integrating with Entropy can pass these fees along

to their users. Whenever a user submits a transaction that requests a

random number, the user can directly pay the entropy fees required with

the native blockchain token. There are no fees for revealing the random

numbers.

You can check the [current

fees](https://docs.pyth.network/entropy/current-fees) page to see the

latest fees for each blockchain.