Get a Random Number

You are viewing the VRF v2 guide - Direct funding method.

To learn how to request random numbers with a subscription, see theSubscription Method guide.

Security Considerations

Be sure to review your contracts with thesecurity considerations in mind.

This guide explains how to get random values using a simple contract to request and receive random values from Chainlink VRF v2 without managing a subscription. To explore more applications of VRF, refer to ourblog .

Requirements

This guide assumes that you know how to create and deploy smart contracts on Ethereum testnets using the following tools:

- The Remix IDE
- MetaMask
- Sepolia testnet ETH

If you are new to developing smart contracts on Ethereum, see theGetting Started guide to learn the basics.

Create and deploy a VRF v2 compatible contract

For this example, use theVRFv2DirectFundingConsumer.sol sample contract. This contract imports the following dependencies:

- VRFV2WrapperConsumerBase.sol(link)
- ConfirmedOwner.sol(link)

The contract also includes pre-configured values for the necessary request parameters such ascallbackGasLimit,requestConfirmations, the number of random wordsnumWords, the VRF v2 Wrapper addresswrapperAddress, and the LINK token addresslinkAddress. You can change these parameters if you want to experiment on different testnets.

Build and deploy the contract on Sepolia.

1. Open theVRFv2DirectFundingConsumer.solcontract in Remix.

Open in Remix What is Remix? 2. On theCompiletab in Remix, compile theVRFv2DirectFundingConsumercontract. 3. Configure your deployment. On theDeploytab in Remix, select theInjected Web3 Environmentand select theVRFv2DirectFundingConsumercontract from the contract list. 4. Click theDeploybutton to deploy your contract onchain. MetaMask opens and asks you to confirm the transaction. 5. After you deploy your contract, copy the address from theDeployed Contractslist in Remix. Before you can request randomness from VRF v2, you must fund your consuming contract with enough LINK tokens in order to request for randomness. Next,fund your contract .

Fund Your Contract

Requests for randomness will fail unless your consuming contract has enough LINK.

1. Acquire testnet LINK .
2. Fund your contract with testnet LINK . For this example, funding your contract with 2 LINK should be sufficient.

Request random values

The deployed contract requests random values from Chainlink VRF, receives those values, builds a structRequestStatuscontaining them, and stores the struct in a mappings_requests. Run therequestRandomWords()function on your contract to start the request.

1. Return to Remix and view your deployed contract functions in theDeployed Contractslist.
2. Click therequestRandomWords()function to send the request for random values to Chainlink VRF. MetaMask opens and asks you to confirm the transaction.

Set your gas limit in MetaMask

Remix IDE doesn't set the right gas limit, so you mustedit the gas limit in MetaMaskwithin theAdvanced gas controlssettings.

For this example to work, set the gas limit to400,000in MetaMask.

First,enableAdvanced gas controlsin your MetaMask settings .

Before confirming your transaction in MetaMask, navigate to the screen where you can edit the gas limit: SelectSite suggested>Advanced>Advanced gas controlsand selectEditnext to theGas limitamount. Update theGas limitvalue to400000and selectSave. Finally, confirm the transaction.After you approve the transaction, Chainlink VRF processes your request. Chainlink VRF fulfills the request and returns the random values to your contract in a callback to thefulfillRandomWords()function. At this point, a new keyrequestIdis added to the mappings_requests. Depending on current testnet conditions, it might take a few minutes for the callback to return the requested random values to your contract. 3. To fetch the request ID of your request, calllastRequestId(). 4. After the oracle returns the random values to your contract, the mappings_requestsis updated. The received random values are stored ins_requests[_requestId].randomWords. 5. CallgetRequestStatus()and specify therequestIdto display the random words.

Note on Requesting Randomness

Do not re-request randomness. For more information, see theVRF Security Considerations page.

Analyzing the contract

In this example, the consuming contract uses static configuration parameters.

// SPDX-License-Identifier: MIT// An example of a consumer contract that directly pays for each request.pragmasolidity^0.8.7;import{ConfirmedOwner}from"@chainlink/contracts/src/v0.8/shared/access/ConfirmedOwner.sol";import{VRFV2WrapperConsumerBase}from"@chainlink/contracts/src/v0.8/ *** Request testnet LINK and ETH here: https://faucets.chain.link/ * Find information on LINK Token Contracts and get the latest ETH and LINK faucets here: https://docs.chain.link/docs/link-token-contracts/** *// * THIS IS AN EXAMPLE CONTRACT THAT USES HARDCODED VALUES FOR CLARITY. * THIS IS AN EXAMPLE CONTRACT THAT USES UN-AUDITED CODE. * DO NOT USE THIS CODE IN PRODUCTION. */contractVRFv2DirectFundingConsumerisVRFV2WrapperConsumerBase,ConfirmedOwner{eventRequestSent(uint256requestId,uint32numWords);eventRequestFulfilled(uint256requestId,uint256[]randomWords,uint256payment);// amount paid in linkboolfulfilled;// whether the request has been successfully fulfilleduint256[]randomWords;}mapping(uint256=>RequestStatus)publics_requests;/ requestId --> requestStatus /// past requests Id.uint256[]publicrequestIds;uint256publiclastRequestId;// Depends on the number of requested values that you want sent to the// fulfillRandomWords() function. Test and adjust// this limit based on the network that you select, the size of the request,// and the processing of the callback request in the fulfillRandomWords()// function.uint32callbackGasLimit=100000;// The default is 3, but you can set this higher.uint16requestConfirmations=3;// For this example, retrieve 2 random values in one request.// Cannot exceed VRFV2Wrapper.getConfig().maxNumWords.uint32numWords=2;// Address LINK - hardcoded for SepoliaaddresslinkAddress=0x779877A7B0D9E8603169DdbD7836e478b4624789;// address WRAPPER - hardcoded for SepoliaaddresswrapperAddress=0xab18414CD93297B0d12ac29E63Ca20f515b3DB46;constructor()ConfirmedOwner(msg.sender)VRFV2WrapperConsumerBase(linkAddress,wrapperAddress){}functionrequestRandomWords()externalonlyOwnerreturns(uint256requestId){requestId=requestRandomness(callbackGasLimit,requestConfirmations,numWords);s_requests[requestId]=RequestStatus({paid:VRF_V2_WRAPPER.calculateRequestPrice(callbackGasLimit),randomWords:newuint256[](0),fulfilled:false});requestIds.push(requestId);lastRequestId=requestId;emitRequestSent(requestId,numWords);returnrequestId;}functionfulfillRandomWords(uint256_requestId,uint256[]memory_randomWords)internal{require(s_requests[_requestId].paid>0,"request not found");s_requests[_requestId].fulfilled=true;s_requests[_requestId].randomWords=_randomWords;emitRequestFulfilled(_requestId,_randomWords,s_requests[_requestId].paid);}functiongetRequestStatus(uint256_requestId)externalview{require(s_requests[_requestId].paid>0,"request not found");RequestStatusmemoryrequest=s_requests[_requestId];return(request.paid,request.fulfilled,request.randomWords);}/ * Allow withdraw of Link tokens from the contract /functionwithdrawLink()publiconlyOwner{LinkTokenInterface link=LinkTokenInterface(linkAddress);require(link.transfer(msg.sender,link.balanceOf(address(this))),"Unable to transfer");}} Open in Remix What is Remix? The parameters define how your requests will be processed. You can find the values for your network in theSupported networks page.

- uint32 callbackGasLimit: The limit for how much gas to use for the callback request to your contract'sfulfillRandomWords()function. It must be less than themaxGasLimitlimit on the coordinator contract minus thewrapperGasOverhead. See theVRF v2 Direct funding limits for more details. Adjust this value for larger requests depending on how yourfulfillRandomWords()function processes and stores the received random values. If yourcallbackGasLimitis not sufficient, the callback will fail and your consuming contract is still charged for the work done to generate your requested random values.
- uint16 requestConfirmations: How many confirmations the Chainlink node should wait before responding. The longer the node waits, the more secure the random value is. It must be greater than theminimumRequestBlockConfirmationslimit on the coordinator contract.

- uint32 numWords: How many random values to request. If you can use several random values in a single callback, you can reduce the amount of gas that you spend per random value. The total cost of the callback request depends on how yourfulfillRandomWords()function processes and stores the received random values, so adjust yourcallbackGasLimitaccordingly.

The contract includes the following functions:

- requestRandomWords(): Takes your specified parameters and submits the request to the VRF v2 Wrapper contract.
- fulfillRandomWords(): Receives random values and stores them with your contract.
- getRequestStatus(): Retrive request details for a given_requestId.
- withdrawLink(): At any time, the owner of the contract can withdraw outstanding LINK balance from it.

Security Considerations

Be sure to review your contracts to make sure they follow the best practices on theecurity considerations page.

Clean up

After you are done with this contract, you can retrieve the remaining testnet LINK to use with other examples.

1. CallwithdrawLink()function. MetaMask opens and asks you to confirm the transaction. After you approve the transaction, the remaining LINK will be transferred from your consuming contract to your wallet address.