Get a Random Number

You are viewing the VRF v2 guide - Subscription method

To learn how to request random numbers without a subscription, see thedirect funding 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. For more advanced examples with programmatic subscription configuration, see thProgrammatic Subscription page. To explore more applications of VRF, refer to oublog .

Subscription Manager

Read theSubscription Manager UI page to learn how to use all the features of the VRF v2 user interface. To learn how to troubleshoot your VRF requests, read thpending andfailed requests sections.

Go tovrf.chain.link to open the Subscription Manager.

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 fund a subscription

For this example, create a new subscription on the Sepolia testnet.

1. Open MetaMask and set it to use the Sepolia testnet. TheSubscription Manager detects your network based on the active network in MetaMask.
2. Check MetaMask to make sure you have testnet ETH and LINK on Sepolia. You can get testnet ETH and LINK afaucets.chain.link .
3. Open the Subscription Manager atvrf.chain.link .

Open the Subscription Manager 4. ClickCreate Subscriptionand follow the instructions to create a new subscription account. If you connect your wallet to the Subscription Manager, theAdmin addressfor your subscription is prefilled and not editable. Optionally, you can enter an email address and a project name for your subscription, and both of these are private. MetaMask opens and asks you to confirm payment to create the account onchain. After you approve the transaction, the network confirms the creation of your subscription account onchain. 5. After the subscription is created, clickAdd fundsand follow the instructions to fund your subscription.

- For your request to go through, you need to fund your subscription with enough LINK to meet yourminimum subscription balance to serve as a buffer against gas volatility. For this example, a balance of 12 LINK is sufficient. (After your request is processed, it costs around 3 LINK, and that amount will be deducted from your subscription balance.)
- MetaMask opens to confirm the LINK transfer to your subscription. After you approve the transaction, the network confirms the transfer of your LINK token to your subscription account.
- After you add funds, clickAdd consumer. A page opens with your account details and subscription ID.
- Record your subscription ID, which you need for your consuming contract. You will add the consuming contract to your subscription later.

You can always find your subscription IDs, balances, and consumers atvrf.chain.link .

Now that you have a funded subscription account and your subscription IDcreate and deploy a VRF v2 compatible contract.

Create and deploy a VRF v2 compatible contract

For this example, use theVRFv2Consumer.sol sample contract. This contract imports the following dependencies:

- VRFConsumerBaseV2.sol(link)
- VRFCoordinatorV2Interface.sol(link)
- ConfirmedOwner.sol(link)

The contract also includes pre-configured values for the necessary request parameters such asvrfCoordinatoraddress, gas lanekeyHash,callbackGasLimit,requestConfirmationsand number of random wordsnumWords. You can change these parameters if you want to experiment on different testnets, but for this example you only need to specifysubscriptionIdwhen you deploy the contract.

Build and deploy the contract on Sepolia.

1. Open theVRFv2Consumer.solcontract in Remix.

Open in Remix What is Remix? 2. On theCompiletab in Remix, compile theVRFv2Consumer.solcontract. 3. Configure your deployment. On theDeploytab in Remix, select theInjected Providerenvironment, select theVRFv2Consumercontract from the contract list, and specify yoursubscriptionIdso the constructor can set it. 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 add this address as an approved consuming contract on your subscription account. 6. Open the Subscription Manager atvrf.chain.link and click the ID of your new subscription under theMy Subscriptionslist. The subscription details page opens. 7. Under theConsumerssection, clickAdd consumer. 8. Enter the address of your consuming contract that you just deployed and clickAdd consumer. MetaMask opens and asks you to confirm the transaction.

Your example contract is deployed and approved to use your subscription balance to pay for VRF v2 requests. Nextrequest random values from Chainlink VRF.

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. 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. You can see a list of pending requests for your subscription ID atvrf.chain.link . 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()specifying therequestIdto display the random words.

You deployed a simple contract that can request and receive random values from Chainlink VRF. To see more advanced examples where the contract can complete the entire process including subscription setup and management, see thProgrammatic Subscription page.

Note on Requesting Randomness

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

Analyzing the contract

In this example, your MetaMask wallet is the subscription owner and you created a consuming contract to use that subscription. The consuming contract uses static configuration parameters.

// SPDX-License-Identifier: MIT// An example of a consumer contract that relies on a subscription for funding.pragmasolidity^0.8.7;import{VRFCoordinatorV2Interface}from"@chainlink/contracts/src/v0.8/interfaces/VRFCoordinatorV2Interface.sol";import{VRFConsumerBaseV2}from"@chainlink/contracts/s *** 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.** */contractVRFv2ConsumerisVRFConsumerBaseV2,ConfirmedOwner{eventRequestSent(uint256requestId,uint32numWords);eventRequestFulfilled(uint256requestId,uint256[]randomWords);structReques whether the request has been successfully fulfilledboolexists;// whether a requestId existsuint256[]randomWords;}mapping(uint256=>RequestStatus)publics_requests;/* requestId --> requestStatus /VRFCoordinatorV2Interface COORDINATOR;// Your subscription ID.uint64s_subscriptionId;// past requests Id.uint256[]publicrequestIds;uint256publiclastRequestId;// The gas lane to use, which specifies the maximum gas price to bump to.// For a list of available gas lanes on each network,// see https://docs.chain.link/docs/vrf/v2/subscription/supported-

*networks/#configurationsbytes32keyHash=0x474e34a077df58807dbe9c96d3c009b23b3c6d0cce433e59bbf5b34f823bc56c;// Depends on the number of requested values that you want sent to the// fulfillRandomWords() function. Storing each word costs about 20,000 gas,// so 100,000 is a safe default for this example contract. 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 VRFCoordinatorV2.MAX_NUM_WORDS.uint32numWords=2;/ * HARDCODED FOR SEPOLIA * COORDINATOR: 0x8103B0A8A00be2DDC778e6e7eaa21791Cd364625*
/constructor(uint64subscriptionId)VRFConsumerBaseV2(0x8103B0A8A00be2DDC778e6e7eaa21791Cd364625)ConfirmedOwner(msg.sender)
{COORDINATOR=VRFCoordinatorV2Interface(0x8103B0A8A00be2DDC778e6e7eaa21791Cd364625);s_subscriptionId=subscriptionId;}// Assumes the subscription is funded sufficiently.functionrequestRandomWords()externalonlyOwnerreturns(uint256requestId){// Will revert if subscription is not set and funded.requestId=COORDINATOR.requestRandomWords(keyHash,s_subscriptionId,requestConfirmations,callbackGasLimit,numWords);s_requests[requestId]=RequestStatus({randomWords:newuint25
,exists:true,fulfilled:false});requestIds.push(requestId);lastRequestId=requestId;emitRequestSent(requestId,numWords);returnrequestId;}functionfulfillRandomWords(uint256_requestId,uint256[]memory_r
not
found");s_requests[_requestId].fulfilled=true;s_requests[_requestId].randomWords=_randomWords;emitRequestFulfilled(_requestId,_randomWords);}functiongetRequestStatus(uint256_requestId)externa
{require(s_requests[_requestId].exists,"request not found");RequestStatusmemoryrequest=s_requests[_requestId];return(request.fulfilled,request.randomWords);}} Open in Remix What is Remix? The parameters define how your requests will be processed. You can find the values for your network in theConfiguration page.

- uint64 s_subscriptionId: The subscription ID that this contract uses for funding requests.
- bytes32 keyHash: The gas lane key hash value, which is the maximum gas price you are willing to pay for a request in wei. It functions as an ID of the offchain VRF job that runs in response to requests.
- 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. 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 subscription 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 coordinator contract.
- fulfillRandomWords(): Receives random values and stores them with your contract.
- getRequestStatus(): Retrive request details for a given_requestId.

Security Considerations

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

Clean up

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

1. Open the Subscription Manager atvrf.chain.link and click the ID of your new subscription under theMy Subscriptionslist. The subscription details page opens.
2. On your subscription details page, expand theActionsmenu and selectCancel subscription. A field displays, prompting you to add the wallet address you want to send the remaining funds to.
3. Enter your wallet address and clickCancel subscription. MetaMask opens and asks you to confirm the transaction. After you approve the transaction, Chainlink VRF closes your subscription account and sends the remaining LINK to your wallet.

Vyper example

You must import theVRFCoordinatorV2Vyper interface. You can find ithere . You can find aVRFConsumerV2examplehere . Read theapeworx-starter-kitREADME to learn how to run the example.