VRF Best Practices

Security Considerations

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

These are example best practices for using Chainlink VRF. To explore more applications of VRF, refer to ourblog .

Getting a random number within a range

If you need to generate a random number within a given range, usemodulo to define the limits of your range. Below you can see how to get a random number in a range from 1 to 50.

functionfulfillRandomWords(uint256,/ *requestId* /uint256[]memoryrandomWords)internaloverride{// Assuming only one random word was requested.s_randomRange=(randomWords[0]%50)+1;}

Getting multiple random values

If you want to get multiple random values from a single VRF request, you can request this directly with thenumWordsargument:

- If you are using the VRF v2 subscription method, see theGet a Random Number guide for an example where one request returns multiple random values.
- If you are using the VRF v2 direct funding method, see theGet a Random Number guide for an example where one request returns multiple random values.

Processing simultaneous VRF requests

If you want to have multiple VRF requests processing simultaneously, create a mapping betweenrequestIdand the response. You might also create a mapping between therequestIdand the address of the requester to track which address made each request.

mapping(uint256=>uint256[])publics_requestIdToRandomWords;mapping(uint256=>address)publics_requestIdToAddress;uint256publics_requestId;functionrequestRandomWords()externalonlyOwnerret
{uint256requestId=COORDINATOR.requestRandomWords(keyHash,s_subscriptionId,requestConfirmations,callbackGasLimit,numWords);s_requestIdToAddress[requestId]=msg.sender;// Store the
latest requestId for this example.s_requestId=requestId;// Return the requestId to the
requester.returnrequestId;}functionfulfillRandomWords(uint256requestId,uint256[]memoryrandomWords)internaloverride{// You can return the value to the requester,// but this example simply stores
it.s_requestIdToRandomWords[requestId]=randomWords;} You could also map therequestIdto an index to keep track of the order in which a request was made.

mapping(uint256=>uint256)s_requestIdToRequestIndex;mapping(uint256=>uint256[])publics_requestIndexToRandomWords;uint256publicrequestCounter;functionrequestRandomWords()externalonlyOv

Processing VRF responses through different execution paths

If you want to process VRF responses depending on predetermined conditions, you can create anenum. When requesting for randomness, map eachrequestIdto an enum. This way, you can handle different execution paths infulfillRandomWords. See the following example:

// SPDX-License-Identifier: MIT// An example of a consumer contract that relies on a subscription for funding.// It shows how to setup multiple execution paths for handling a
response.pragmasolidity^0.8.7;import{VRFCoordinatorV2Interface}from"@chainlink/contracts/src/v0.8/vrf/interfaces/VRFCoordinatorV2Interface.sol";import{VRFConsumerBaseV2}from"@chainlink/contra
** 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.* /contractVRFv2MultiplePathsisVRFConsumerBaseV2{VRFCoordinatorV2Interface COORDINATOR;// Your subscription ID.uint64s_subscriptionId;// Sepolia coordinator. For other
networks,// see https://docs.chain.link/docs/vrf/v2/supported-networks/#configurationsaddressvrfCoordinator=0x8103B0A8A00be2DDC778e6e7eaa21791Cd364625;// 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/supported-
networks/#configurationsbytes32keyHash=0x474e34a077df58807dbe9c96d3c009b23b3c6d0cce433e59bbf5b34f823bc56c;uint32callbackGasLimit=100000;// The default is 3, but you can set this
higher.uint16requestConfirmations=3;// For this example, retrieve 1 random value in one request.// Cannot exceed
VRFCoordinatorV2.MAX_NUM_WORDS.uint32numWords=1;enumVariable{A,B,C}uint256publicvariableA;uint256publicvariableB;uint256publicvariableC;mapping(uint256=>Variable)publicrequests;//
eventseventFulfilledA(uint256requestId,uint256value);eventFulfilledB(uint256requestId,uint256value);eventFulfilledC(uint256requestId,uint256value);constructor(uint64subscriptionId)VRFConsumerBase
{COORDINATOR=VRFCoordinatorV2Interface(vrfCoordinator);s_subscriptionId=subscriptionId;}functionupdateVariable(uint256input)public{uint256requestId=COORDINATOR.requestRandomWords(ke
{requests[requestId]=Variable.A;}elseif(input%3==0)
{requests[requestId]=Variable.B;}else{requests[requestId]=Variable.C;}}functionfulfillRandomWords(uint256requestId,uint256[]memoryrandomWords)internaloverride{Variable
variable=requests[requestId];if(variable==Variable.A){fulfillA(requestId,randomWords[0]);}elseif(variable==Variable.B){fulfillB(requestId,randomWords[0]);}elseif(variable==Variable.C)
{fulfillC(requestId,randomWords[0]);}}functionfulfillA(uint256requestId,uint256randomWord)private{// execution path
AvariableA=randomWord;emitFulfilledA(requestId,randomWord);}functionfulfillB(uint256requestId,uint256randomWord)private{// execution path
BvariableB=randomWord;emitFulfilledB(requestId,randomWord);}functionfulfillC(uint256requestId,uint256randomWord)private{// execution path
CvariableC=randomWord;emitFulfilledC(requestId,randomWord);}} Open in Remix What is Remix?