

# VRF Best Practices

## Security Considerations

Be sure to review your contracts with the [security considerations](#) in mind.

These are example best practices for using Chainlink VRF. To explore more applications of VRF, refer to [ourlog](#).

## Getting a random number within a range

If you need to generate a random number within a given range, use [modulo](#) to define the limits of your range. Below you can see how to get a random number in a range from 1 to 50.

```
function fulfillRandomWords(uint256, requestId) uint256[] memory randomWords; internal override { // 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 the `numWords` argument:

- If you are using the VRF v2 subscription method, see the [Get 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 the [Get 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 between `requestId` and the response. You might also create a mapping between `requestId` and the address of the requester to track which address made each request.

```
mapping(uint256 => uint256[]) public _requestIdToRandomWords; mapping(uint256 => address) public _requestIdToAddress; uint256 public _requestId; function requestRandomWords() external onlyOwner { // requestId = 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. return requestId; } function fulfillRandomWords(uint256 requestId, uint256[] memory randomWords) internal override { // You can return the value to the requester, // but this example simply stores it. s_requestIdToRandomWords[_requestId] = randomWords; } You could also map therequestId to an index to keep track of the order in which a request was made.
```

```
mapping(uint256 => uint256) s_requestIdToRequestIndex; mapping(uint256 => uint256[]) public _requestIndexToRandomWords; uint256 public requestCounter; function requestRandomWords() external onlyOwner { }
```

## Processing VRF responses through different execution paths

If you want to process VRF responses depending on predetermined conditions, you can create an enum. When requesting for randomness, map each `requestId` to an enum. This way, you can handle different execution paths in `fulfillRandomWords`. 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. pragma solidity ^0.8.7; import {VRFCoordinatorV2Interface} from "@chainlink/contracts/src/v0.8/interfaces/VRFCoordinatorV2Interface.sol"; import {VRFConsumerBaseV2} from "@chainlink/contracts/v0.8/src/consumer/VRFConsumerBaseV2.sol"; * 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. /contract VRFv2MultiplePaths is VRFConsumerBaseV2 { VRFCoordinatorV2Interface COORDINATOR; // Your subscription ID. uint64 s_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; uint32 callbackGasLimit=100000; // The default is 3, but you can set this higher. uint16 requestConfirmations=3; // For this example, retrieve 1 random value in one request. // Cannot exceed VRFCoordinatorV2.MAX_NUM_WORDS. uint32 numWords=1; enum Variable { A, B, C } uint256 public variableA; uint256 public variableB; uint256 public variableC; mapping(uint256 => Variable) public requests; // event eventFulfilledA(uint256 requestId, uint256 value); event FulfilledB(uint256 requestId, uint256 value); event FulfilledC(uint256 requestId, uint256 value); constructor(uint64 subscriptionId) VRFConsumerBaseV2(COORDINATOR=VRFCoordinatorV2Interface(vrfCoordinator); s_subscriptionId=subscriptionId; function updateVariable(uint256 input) public { uint256 requestId = COORDINATOR.requestRandomWords(keyHash, s_subscriptionId, requestConfirmations, callbackGasLimit, numWords); if (input % 3 == 0) { requests[_requestId] = Variable.A; } else if (input % 3 == 1) { requests[_requestId] = Variable.B; } else { requests[_requestId] = Variable.C; } } function fulfillRandomWords(uint256 requestId, uint256[] memory randomWords) internal override { Variable variable = requests[_requestId]; if (variable == Variable.A) { fulfillA(requestId, randomWords[0]); } else if (variable == Variable.B) { fulfillB(requestId, randomWords[0]); } else if (variable == Variable.C) { fulfillC(requestId, randomWords[0]); } } function fulfillA(uint256 requestId, uint256 randomWord) private { // execution path variableA = randomWord; emit FulfilledA(requestId, randomWord); } function fulfillB(uint256 requestId, uint256 randomWord) private { // execution path variableB = randomWord; emit FulfilledB(requestId, randomWord); } function fulfillC(uint256 requestId, uint256 randomWord) private { // execution path variableC = randomWord; emit FulfilledC(requestId, randomWord); } } Open in Remix What is Remix?
```