VRF Security Considerations

Gaining access to high quality randomness onchain requires a solution like Chainlink's VRF, but it also requires you to understand some of the ways that miners or validators can potentially manipulate randomness generation. Here are some of the top security considerations you should review in your project.

UserequestIdto match randomness requests with their fulfillment in order

If your contract could have multiple VRF requests in flight simultaneously, you must ensure that the order in which the VRF fulfillments arrive cannot be used to manipulate your contract's user-significant behavior.

Blockchain miners/validators can control the order in which your requests appear onchain, and hence the order in which your contract responds to them.

For example, if you made randomness requestsA,B,Cin short succession, there is no guarantee that the associated randomness fulfillments will also be in orderA,B,C. The randomness fulfillments might just as well arrive at your contract in orderC,A,Bor any other order.

We recommend using therequestIDto match randomness requests with their corresponding fulfillments.

Choose a safe block confirmation time, which will vary between blockchains

In principle, miners/validators of your underlying blockchain could rewrite the chain's history to put a randomness request from your contract into a different block, which would result in a different VRF output. Note that this does not enable a miner to determine the random value in advance. It only enables them to get a fresh random value that might or might not be to their advantage. By way of analogy, they can only re-roll the dice, not predetermine or predict which side it will land on.

You must choose an appropriate confirmation time for the randomness requests you make. Confirmation time is how many blocks the VRF service waits before writing a fulfillment to the chain to make potential rewrite attacks unprofitable in the context of your application and its value-at-risk.

Do not re-request randomness

Any re-request of randomness is an incorrect use of VRFv2. Doing so would give the VRF service provider the option to withhold a VRF fulfillment if the outcome is not favorable to them and wait for the re-request in the hopes that they get a better outcome, similar to the considerations with block confirmation time.

Re-requesting randomness is easily detectable onchain and should be avoided for use cases that want to be considered as using VRFv2 correctly.

Don't accept bids/bets/inputs after you have made a randomness request

Consider the example of a contract that mints a random NFT in response to a user's actions.

The contract should:

1. Record whatever actions of the user may affect the generated NFT.
2. Stop accepting further user actions that might affect the generated NFTand issue a randomness request.
3. On randomness fulfillment, mint the NFT.

Generally speaking, whenever an outcome in your contract depends on some user-supplied inputs and randomness, the contract should not accept any additional user-supplied inputs after it submits the randomness request.

Otherwise, the cryptoeconomic security properties may be violated by an attacker that can rewrite the chain.

fulfillRandomWordsmust not revert

If yourfulfillRandomWords()implementation reverts, the VRF service will not attempt to call it a second time. Make sure your contract logic does not revert. Consider simply storing the randomness and taking more complex follow-on actions in separate contract calls made by you, your users, or anAutomation Node .

UseVRFConsumerBaseV2in your contract, to interact with the VRF service

If you implement thesubscription method , useVRFConsumerBaseV2. It includes a check to ensure the randomness is fulfilled byVRFCoordinatorV2. For this reason, it is a best practice to inherit fromVRFConsumerBaseV2. Similarly, don't overriderawFulfillRandomness.

UseVRFv2WrapperConsumer.solin your contract, to interact with the VRF service

If you implement thedirect funding method , useVRFv2WrapperConsumer. It includes a check to ensure the randomness is fulfilled by theVRFV2Wrapper. For this reason, it is a best practice to inherit fromVRFv2WrapperConsumer. Similarly, don't overriderawFulfillRandomWords.