Randomness
Since all blockchains are fundamentally deterministic and most are public ledgers, generating truly unpredictatable randomness on-chain presents a challenge. This randomness is crucial for fair outcomes in gaming, lotteries, and unique generation of NFTs. To address this, verifiable random functions (VRFs) provided by oracles offer a solution. VRFs guarantee that the randomness can't be predicted or tampered with, ensuring trust and transparency in these applications.
Overview on VRFs
Pseudo-random but secure: VRFs use a secret key and a nonce (a unique input) to generate an output that appears random. While technically 'pseudo-random', it's practically impossible for another party to predict the outcome without knowing the secret key.
Verifiable output: VRFs produce not only the random number but also a proof that anyone can use to independently verify that the result was generated correctly according to the function's parameters.
Generating Randomness with Pragma
[Pragma](https://www.pragma.build/), an oracle on Starknet provides a solution for generating random numbers using VRFs.
Let's dive into how to use Pragma VRF to generate a random number in a simple dice game contract.
Add Pragma as a Dependency
Edit your cairo project's `Scarb.toml` file to include the path to use Pragma.
```toml
[dependencies]
pragma_lib = { git = "https://github.com/astraly-labs/pragma-lib" }
```
Interface
```cairo,noplayground
{{#include ../listings/ch16-building-advanced-starknet-smart-contracts/
listing_06_dice_game_vrf/src/lib.cairo:interfaces}}
```
{{#label pragma_vrf_interface}}
Listing {{#ref pragma_vrf_interface}} shows a contract interfaces for Pragma VRF and a simple dice game.
Description of Key IPragmaVRF Entrypoints and Their Inputs
The function `request_randomness_from_pragma` initiates a request for verifiable randomness from the Pragma oracle. It does this by emitting an event that triggers the following actions off-chain:
1. Randomness generation: The oracle generates random values and a corresponding proof.
2. On-chain submission: The oracle submits the generated randomness and proof back to the blockchain via the `receive_random_words` callback function.
`request_randomness_from_pragma` Inputs
1. `seed`: A value used to initialize the randomness generation process. This should be unique to ensure unpredictable results.
2. `callback_address`: The contract address where the `receive_random_words`

function will be called to deliver the generated randomness. It is typically the address of your deployed contract implementing Pragma VRF.
3. `callback_fee_limit`: The maximum amount of gas you're willing to spend on executing the `receive_random_words` callback function.
4. `publish_delay`: The minimum delay (in blocks) between requesting randomness and the oracle fulfilling the request.
5. `num_words`: The number of random values (each represented as a `felt252`) you want to receive in a single callback.
6. `calldata`: Additional data you want to pass to the `receive_random_words` callback function.
`receive_randomn_words` Inputs
1. `requester_address`: The contract address that initiated the randomness request.
2. `request_id`: A unique identifier assigned to the randomness request.
3. `random_words`: An array (span) of the generated random values (represented as `felt252`).
4. `calldata`: Additional data passed along with the initial randomness request.
Dice Game Contract
This dice game contract allows players to guess a number between 1 & 6 during an active game window. The contract owner then has the ability to toggle the game window to disable new guesses from players. To determine the winning number, the contract owner calls the `request_randomness_from_pragma` function to request a random number from the Pragma VRF oracle. Once the random number is received through the `receive_random_words` callback function, it is stored in the `last_random_number` storage variable. Each player has to call `process_game_winners` function to determine if they have won or lost. The `last_random_number` generated is then reduced to a number between 1 & 6, and compared to the guesses of the players stored in the `user_guesses` mapping, which leads to the emission of an event `GameWinner` or `GameLost`.
```cairo,noplayground
{{#include ../listings/ch16-building-advanced-starknet-smart-contracts/
listing_06_dice_game_vrf/src/lib.cairo:dice_game}}
```
{{#label dice_game_vrf}}
Listing {{#ref dice_game_vrf}}: Simple Dice Game Contract using Pragma VRF.
NB: Fund Your Contract After Deployment to Utilize Pragma VRF
After deploying your contract that includes Pragma VRF functionalities, ensure it holds sufficient ETH to cover the expenses related to requesting random values. Pragma VRF requires payment for both generating the random numbers and executing the callback function defined in your contract.
For more information, please refer to the [Pragma][pragma] docs.
[pragma]: https://docs.pragma.build/Resources/Starknet/randomness/randomness