Introduction

Band Verifiable Random Function (VRF)

The foundation of the randomness matters for security reasons. Modern distributed applications (dApps) often rely on "good" randomness, generated freshly and independently of the application's state. In addition, the random result should have integrity, which means it should be verifiable, tamper-proof and unpredictable. For example, consider an online lottery in which participants place their bids, a random result comes up, and winnings are awarded according to the bid placement. Similarly, consider a leadership election (often used in committee-based blockchain platforms) that proceeds in rounds to randomly elect a leader among a set of participants. For such applications, it is crucial to guarantee that the randomness is sampled uniformly and independently of the application’s state, therefore making it hard to predict as well as ensuring that there are no malicious actors who can affect the outcome of such lottery winnings and leadership elections.

Verifiable Random Function (VRF)

A commonly used technique for applications that require ”good” random values to be produced rely on cryptography to produce values that are pseudorandom, i.e., impossible to distinguish from uniformly randomized values. A verifiable random function (VRF) is a mathematical operation that takes some input and produces a pseudorandom output along with the proof of authenticity of the output generation process. Challengers can verify the proof to ensure the pseudorandom result is valid and does not come out of thin air.

In general, the core of the VRF system will have a set of secret keys used to generate verifiable results and a set of corresponding public keys used to verify the results produced. For example, given a secret key generated in private and a cryptographic function that maps a seed to an output value with its proof. The crucial property is that someone that does not have access to the secret key cannot distinguish in polynomial time the output from a value that is sampled uniformly at random from the range of all possible outcomes.

VRF security properties

- Unpredictability
- : This ensures that the computed outputs are distributed in a way that is, for all practical purposes, uniformly random. It is a fundamental property of a VRF, as it says that the VRF behaves like a random oracle. In practice, this implies that anyone not knowing the secret key has no way to predict the outcome that is better than “randomly guessing” even when knowing the seed. So, if the input seeds are chosen with sufficiently high entropy, it is practically impossible to predict the output.
- Uniqueness
- : This ensures that, after the VRF providers publish their secret key, they can only produce proof that will convince others of the correct VRF output value for every seed. In other words, for a given (secret key, seed), it is incredibly hard to find two different VRF values, both of which pass the verification. This property is crucial to protect against a cheating actor that tries to claim a specific output other than the correct one for their purposes.
- Collision-Resistance
- : This ensures that it is computationally hard to find two different inputs, "seed1" and "seed2", with the same secret key to obtain the identical output value —much like the classic property of cryptographic hash functions. The difference is that for VRFs, this holds even against an adversary that knows the secret key. Note that this offers a different type of protection than the unique property. For example, it protects against a party that tries to claim an output computed from one input seed as if it was computed from a different "seed2".

Band VRF

Our solution for verifiably (pseudo-)randomness is based on the BandChain blockchain. Our protocol uses a verifiable random function (VRF) to cryptographically secure that produced results have not been tampered with. We will present in detail how our protocol operates in this document.

Band VRF extends from the general form of the VRF system to serve randomness requests for dApps, which is based on the distributed BandChain oracle network. BandChain is a public blockchain that provides APIs for data and services stored “off-chain” on the traditional web or from third-party providers. It supports generic data requests from other public blockchains and performs on-chain aggregation via the BandChain oracle scripts. The aggregation process works like a smart contract on the EVM platform, which executes data on-chain to produce the oracle results. The oracle results are also stored on-chain. After that, the results are returned to the calling dApp on the main blockchain (e.g., Ethereum), accompanied by a proof of authenticity via customized one-way bridges (or via an Inter-Blockchain Communication protocol). To guarantee verifiably “good” randomness suitable for security-critical applications, we deploy the cryptographic primitive of verifiable random functions. At a high level, a VRF provides values that are indistinguishable from uniformly random ones and can be verified for their authenticity concerning a pre-published public key.

We chose the VRF of this paper , which has already been adopted in various other protocols. The construction is based on a well-studied cryptographic hardness assumption over prime-order elliptic curve groups. For our instantiation, we chose the widely-used Ed25519 curve that achieves very good performance and has a transparent choice of parameters, as well as the Elligator, for our hash-to-curve installation. Our implementation is fully compliant with theVRF draft standard . Moreover, we implemented all the necessary techniques to achieve full security. For completeness, we include the pseudo-code description of our VRF below.

Protocol Flow

At a high level, our protocol works as follows. First, the VRFProvider contract and the Bridge contract are deployed on the main-chain. The VRFProvider contract is in charge of receiving randomness requests from dApps and contains code that pre-processes the request in order to be ready for submission to the Band side-chain. It also works as the receiving end of the request's result. The Bridge contract, as the name denotes, works as the connecting "bridge" between the two chains in order to validate the latest state of the side-chain and verify that the received results for VRF requests are indeed the ones computed and stored on the BandChain.

A third-party dApp that wishes to request a random value submits its request to the VRFProvider contract, which then prepares the actual VRF input by expanding it into a VRF seed. This is picked up by incentivized actors and/or the Band foundation and is submitted as a VRF request to the BandChain. In particular, a VRF Oracle Script collects this request, maps it to the set of VRF Data Sources that is available to the chain, as well as a number of BandChain Validators. The VRF Oracle Script then randomly assigns it to a VRF Data Source, which in turn passes the request to a corresponding VRF Provider API. The assigned VRF Provider API evaluates the VRF based on the prescribed input using its VRF secret key, and broadcasts the result to the Band network. All chosen validators run the VRF verification algorithm using this provider's public key and, if the verification is successful, the result will be transmitted to the VRF Oracle Script. Finally, after collecting the necessary number of results from the validators, the oracle script accepts the majority as the final result which then becomes part of the BandChain state, which gets included in the next block's computation.

The final result is transmitted back to the main-chain's VRFProvider contract together with a Merkle tree proof for its inclusion on the BandChain's state. This proof is then checked with the Bridge contract. Upon successful checking, the result is returned to the original calling dApp.

Band VRF on EVM

From the protocol flow, we can catagorize the EVM contracts into two parts: the Band-operated contracts and the consumer (third-party) contract. We provide detailed explanations regarding the consumer contract in the VRF Intergration section. In this section, we will focus on the two Band-operated contracts: Bridge and VRFProvider.

Bridge

Conceptually, you can think of the Bridge contract as a generic logic that helps verify the availability of any data stored on the BandChain. In this case, the VRFProvider acts as a Bridge's user who wants to verify the data. In order to verify the data's availability, the Bridge contract contains a set of validators of the BandChain, which is used for signature verification when any external actors relay a BandChain's block. After the block relaying is successful (accumulated power of more than 2/3), the rest is the verification of the actual data (leaf) against the root hash. Finally, the result/leaf is extracted and returned to the caller if the data is successfully verified.

State

// The encoded chain's ID of Band. // This value is only set at the deployment time bytes public encodedChainID;

//=== Set by the owner
===

struct ValidatorWithPower { address addr; uint256 power; }

// Mapping from an address to its voting power. EnumerableMap.AddressToUintMap private validatorPowers;

// The total voting power of active validators currently on duty. uint256 public totalValidatorPower;

//=== Set by the owner
===

//=== Public
===

struct BlockDetail { bytes32 oracleState; uint64 timeSecond; uint32 timeNanoSecondFraction; // between 0 to 10^9 }

// Mapping from block height to the struct that contains block time and hash of "oracle" iAVL Merkle tree. mapping(uint256 => BlockDetail) public blockDetails;

//=== Public
=== Functions

The two key functions of the Bridge contract areupdateValidatorPowers andrelayAndVerify .

TheupdateValidatorPowers function can only be called by the owner of the Bridge contract. The function is used for updating the Bridge contract's validator set in order for the contract's state to be consistent with the actual validator set on the BandChain.

TherelayAndVerify function is a public function for anyone who wants to relay request data from the BandChain into the Bridge contract. As a result, the relayed and verified data can be used safely by those who wish to consume data from Band oracle.

VRFProvider

The VRFProvider contract is a contract that is used for tracking all VRF requests and their status. It is also the gateway for the users/consumers to interact with the protocol.

State

//=== Set by the owner
===

// A reference to the Bridge contract IBridge public bridge;

// An id of the VRF oracle script on Bandchain uint64 public oracleScriptID;

// A minimum number of validators needed to resolve requests. uint8 public minCount;

// A limited number of validators needed to resolve requests. uint8 public askCount;

// A global counter for all tasks known to the VRFProvider // As a new request is made, a new task will be created, and this number will keep increasing. uint64 public taskNonce;

// An amount of minimum fee required to pay the VRF worker(resolver, relayer, etc.). uint256 public minimumFee;

//=== Set by the owner
===

//=== Public
===

// Mapping that enforces the client to provide a unique seed for each request mapping(address => mapping(string => bool)) public hasClientSeed;

// The struct of a task struct Task { bool isResolved; uint64 time; address caller; uint256 taskFee; bytes32 seed; string clientSeed; bytes proof; bytes result; } // Mapping from nonce => task // External entities can query a task via this mapping by specifying the task id. // The task id is the taskNonce at the time the task was created. mapping(uint64 => Task) public tasks;

//=== Public
=== Functions

There are two main functions in the VRFProvider contract:requestRandomData andrelayProof .

TherequestRandomData function is a function that is called by VRF users/consumers. This function is used for registering a new task. It also performs a few checks, such as checking that the users/consumers are not using an already used seed, and that the fee meets the minimum requirement.

TherelayProof function is a function that is called by the VRF resolvers (worker bots, relayers, etc.). This function receives the Band's Merkle proof and then passes it to the Bridge contract for verification. After the verification has succeeded, the extracted result is sent back from the Bridge contract to the VRFProvider contract. The VRFProvider contract then verifies the parameters in the result's struct to check if they pass all conditions. After the result's parameters have been checked, the VRFProvider contract calls back to the consumer contract via the callback function on the consumer side with the final VRF result as one of the parameters.

Learn more

To learn more about the VRFProvider and Bridge, please visit this repository

Band VRF on BandChain

There are two main components on the BandChain which involve in the VRF random data delivery process:oracle script anddata source .

The oracle script on the BandChain is similar to a smart contract on EVM chains or CosmWasm chains; it is the on-chain calculation part. VRF oracle script is an oracle script that has been designed specifically for the VRF protocol. It contains a list of VRF data sources to be chosen from during the preparation stage. After the preparation stage, the chosen validators run the VRF data sources to create the VRF random result, and then report the raw result to the BandChain's state. Finally, at the execution/aggregation stage, the oracle script is run automatically to summarize the raw results into a single result, which is then stored on the BandChain's state. When the final result is stored on the BandChain, anyone can retrieve its Merkle proof of availability from the BandChain. This proof is used together with the relay function of the Bridge contract on the client chain.

The data source is an off-chain execution script that is used to call external APIs. VRF data sources are linked to external VRF providers that provide the verifiable randomness service. After calling the provider, the data source verifies the returned result and proof against the predefined public key. The data source only returns the result if the verification is successful.

Learn more

- Band VRF Whitepaper
- Band oracle script docs
- Band data source docs Previous Oracle WebAssembly (Owasm) Next Getting Started