

# Fast SNARK-based Non-Interactive Distributed Verifiable Random Function with Ethereum Compatibility

Jia Liu   
Enya Labs  
United Kingdom

Mark Manulis   
Universität der Bundeswehr München  
Germany  
mark@manulis.eu

## ABSTRACT

Distributed randomness beacons (DRBs) are fundamental for various decentralised applications, such as consensus protocols, decentralised gaming and lotteries, and collective governance protocols. These applications are heavily used on modern blockchain platforms. This paper presents the so far most efficient direct construction and implementation of a non-interactive distributed verifiable random function (NI-DVRF) that is fully compatible with Ethereum. Our NI-DVRF scheme adopts pairings and combines techniques from secret sharing, SNARKs, and BLS signatures. The security properties of the resulting NI-DVRF scheme are formally modelled and proven in the random oracle model under standard pairing-based assumptions. To justify the efficiency and cost claims and more generally its adoption potential in practice, the proposed NI-DVRF scheme was implemented in Rust and Solidity. Our implementation is highly optimised and is currently being investigated for deployment on the multichain layer-2 scaling solution provided by Boba Network to power its DRB service zkRand. Our experimental analysis, therefore, also evaluates performance and scalability properties of the proposed NI-DVRF and its implementation.

## 1 INTRODUCTION

**DRBs and DVRFs.** Distributed randomness beacons (DRBs) [1], see also a recent survey in [17], enable a group of  $n$  participants to collectively compute a (verifiable) random output without any participant or coalition of participants being able to predict or bias the outcome. This makes DRBs essential for a variety of applications in distributed consensus protocols and blockchain networks, e.g.:

- *Random Sampling in Decentralized Networks:* In decentralized systems, it is often necessary to select a random sample of nodes for certain tasks. Blockchains, for instance, require a reliable source of public randomness for various activities, such as forming a consensus group, shuffling validators, electing proposers for the next block, or assigning tasks in a decentralized network. DRBs can be used to generate random numbers that no participant can predict or influence.
- *Decentralized Lotteries and Gaming:* Fairness in the draw is of utmost importance in decentralized lotteries. An ideal solution would be a distributed randomness beacon that can be verified by anyone, and which no participant can manipulate. This also applies to other forms of gaming where randomness is required.
- *Decentralized Decision Making:* In any decentralized decision-making process, whether it be voting systems or collective governance protocols, it may be necessary to randomly select participants for specific roles, responsibilities, or rights.

A public, verifiable source of randomness would be crucial in these systems for ensuring the fairness of the selection process.

DRB protocols can be constructed from distributed verifiable random functions (DVRFs) [23, 29, 31]. A DVRF is a  $t$ -out-of- $n$  threshold scheme where  $n$  participants first jointly run a distributed key generation (DKG) protocol, e.g. [25], to establish global public parameters and their secret keys (shares). They then use their secret keys to create pseudorandom values repeatedly at pre-defined intervals. A suitable DVRF should at least satisfy the following informal security properties [23]:

- *Uniqueness:* given an input  $x$ , the output pseudorandom value  $v$  is deterministic. Uniqueness is crucial for certain decentralized applications, such as selecting the next block proposer or the next set of validators in a consensus protocol. Uniqueness guarantees a deterministic choice for the next block, and without it, the blockchain will fork and fail to reach consensus.
- *Public verifiability:* anyone can verify a cryptographic proof that the pseudorandom value  $v$  is correctly computed. There is no need to trust any party to honestly create the random value.
- *Pseudo-randomness:* the distribution of  $v$  is indistinguishable from a random. No one can predict the value of  $v$  before it's created, and no one is able to bias the value of  $v$  to their advantage.

When it comes to the adoption of DVRFs for blockchain applications, many practically driven requirements must be considered, in addition to the security properties. It is crucial for DVRFs to not only be efficient and scalable but also to ensure low latencies since distributed blockchain nodes form an ad-hoc network. Non-interactive DVRFs are particularly attractive to blockchain networks because each party only needs to send one message and does not have to wait for messages from other parties to compute their own messages, which greatly reduces latencies and simplifies dispute resolutions. Furthermore, for fast adoption in blockchain industry DVRFs need to be designed and implemented in a way that is compatible with existing blockchain infrastructures. In this work we focus on DVRFs for Ethereum and Ethereum-compatible blockchains, since Ethereum is currently one of the most widely used blockchain platforms.

**Related work on DVRF for blockchains.** Existing DVRF constructions and implementations fall short of simultaneously achieving the required security guarantees and the aforementioned practically driven requirements as we argue in the following. DVRF constructions in [1, 15, 23, 30] rely on an interactive DKG protocol to

initialise the global public parameters and participants’ secret keys. The interactive setup involves several rounds of communications among the participants and a complex dispute resolution process, making it difficult to implement in practice. Intuitively, a more promising approach would be to adopt a non-interactive DKG (NI-DKG), yet existing NI-DKG protocols have a number of drawbacks for such adoption. For example, in the NI-DKG protocol by Groth [29] each participant publishes a huge amount of data, referred to as chunk encryptions for shares and proofs, and its verification time is very slow: the chunk encryption size is  $O(n \cdot m)$  where  $n$  is the total number of participants and  $m$  is the chunk size; the size of a chunk proof is  $O(\ell + n)$  and its verification involves  $O(n \cdot m)$  exponentiations in the pairing input group  $\mathbb{G}_1$ ; the verification of its secret sharing proof involves  $t$  exponentiations in  $\mathbb{G}_2$  and  $n$  exponentiations in  $\mathbb{G}_1$ . The implementation in [29] is not compatible with Ethereum since it uses the BLS12-381 curve instead of BN254 (which is also faster than BLS12-381) and requires exponentiations in  $\mathbb{G}_2$  for verification, which are not supported by Ethereum.

The recent SNARK-based NI-DKG implementation in [22] is the only work, which is conceptually similar to our approach. In comparison to our DVRF protocol, [22] uses special algebraic properties of BLS12-377/BW6 curves, also known as 2-chain pairing curves. However, these curves are not natively supported on Ethereum, as there are no precompiles for curve operations on BLS12-377/BW6. Moreover, [22] provides only a source code for which there exists no formal academic work and hence also no security evaluation.

Recent development in threshold BLS [9, 24] modifies the BLS signature verification process to support non-interactive key setup. However, this modification leads to a loss of uniqueness in the obtained BLS signatures, making them unsuitable for building a DVRF. In fact, exponentially many signatures can pass the modified verification for each message, which would result in a DVRF lacking the uniqueness property.

**Other approaches for building DRB.** As mentioned in [17] there are different approaches for building DRB protocols. For example, in leader-based selection DRB protocols, such as Algorand [27], Ouroboros-Praos [18] and Elrond [20], a single leader node is selected each round to produce the next beacon value using a VRF function [33]. The setup phase of such protocols is simple and non-interactive as each participant chooses their own VRF key pairs. However, this type of protocols is subject to withholding attacks as the leader node might refuse to provide the random value. Another approach, e.g., pursued by SCRAPE [13], RandShare/RandHound [37] and Ouroboros [32], is to directly use the interactive Publicly Verifiable Secret Sharing (PVSS) to generate each random beacon. However, this method is expensive as PVSS has similar communication and computation complexity to an interactive DKG protocol, and as observed in [17], typically results in poorer scalability when compared to DVRF-based approaches.

Commit-reveal [10, 35] is another classic way to construct DRB protocols. In the commit phase, each participant commits a random seed. Once all the participants have shared their commitments, each participant opens its commitment by revealing its random seed. All the random seeds can then be aggregated to create the final output. However, this approach is susceptible to bias as the last revealing participant can compute the final output earlier and decide whether

to disclose its random seed or not. This is precisely the last-revealer attack from [19], which can be mitigated using DKG protocols that can be viewed as a threshold extension of the commit-reveal approach. Another method to mitigate the last-revealer attack is to base the construction of a DRB protocol on a verifiable delay function (VDF) [16, 34, 38], which is a function that always takes a predetermined time to compute. In practice, however, recovering VDF outputs requires a substantial amount of computation such that honest users must employ expensive hardware, such as high-end ASICs, to establish a fast baseline. In fact, delay is not a provable security guarantee and some designs such as Minroot [6] have been broken using parallel computation for acceleration.

**Our contributions.** We focus on a non-interactive DVRF (NI-DVRF) where the distributed key setup as well as the generation of pseudorandom values are both performed in a non-interactive manner; that is, each participant asynchronously sends only one message and there is only one communication round. Our main contributions are:

- We present an efficient direct NI-DVRF construction based on standard pairing assumptions using techniques from secret sharing, SNARKs [21, 28], and BLS signatures [11]. Our NI-DVRF construction improves upon the DVRF protocol from [23], which uses an interactive DKG. By deploying suitable SNARKs, we achieve a completely non-interactive construction that proceeds through two main phases:
  - NI-DKG phase: Each of the  $n$  participants distributes their own secret key using a  $t$ -out-of- $n$  threshold secret sharing scheme in a non-interactive manner. To do this, each participant evaluates a random polynomial at  $n$  points to create  $n$  shares. These shares are then encrypted and distributed to the intended recipients along with a SNARK proof. This proof ensures that the encrypted shares and public parameters are computed correctly and can be publicly verified, achieving non-interactivity, succinct proof size, and fast verification time. This phase is the initial setup phase and is executed only once.
  - Pseudorandom generation: After setting up the keys in the initial NI-DKG phase, participants can collaborate continuously to compute pseudorandom values for many epochs. Each epoch is uniquely identified by a public input, such as a timestamp or counter. Each participant provides a partial evaluation on the input, and a threshold number of these partial evaluations can be combined to produce the final pseudorandom output. The combination of partial evaluations does not involve any secret information, allowing any node in the network to perform it. Therefore, this phase is also non-interactive. Each pseudorandom output is deterministic for each public input, but remains unpredictable until it is generated.
- We provide formal definitions for NI-DVRF and security proofs for both standard and strong pseudorandomness properties [23] of our construction. For standard pseudorandomness, values output by NI-DVRF must remain indistinguishable from uniformly distributed random values,

whereas for strong pseudorandomness the adversary may additionally query the partial evaluation oracle on the challenge public input up to a certain number of times. We prove that our NI-DVRF instantiation achieves standard pseudorandomness under the co-CDH and SDH assumptions, and strong pseudorandomness under co-CDH and XDH assumptions, in the random oracle model.

- We describe the proof-of-concept implementation of our NI-DVRF and provide experimental evaluation of its performance and costs. As mentioned, our key objective is to ensure compatibility with Ethereum, which is one of the most widely used blockchain platforms, we implement our NI-DKG protocol in Halo2 using non-native encoding for BN256 [4, 5], considering Ethereum has precompiles for BN256 curve operations [2]. We stress that proving non-native encoding of operations in SNARKs is typically expensive. Therefore, we introduce several non-trivial optimisations to significantly reduce the peak memory usage and the proving time, making our implementation to be the first that is highly practical *and* compatible with Ethereum. The two main optimisations we have developed are:
  - Our NI-DKG protocol only requires a fixed generator  $g_1 \in \mathbb{G}_1$  for creating public shares and a fixed generator  $g_2 \in \mathbb{G}_2$  for creating global public key on BN256, we have developed windowed scalar multiplication circuits for fixed point generator which reduced more than 70% of gates.
  - The encryption of shares are performed on Grumpkin curve [3] (instead of BN256 [26]) for which we have developed ecc-chip to generate circuits. Since the base field of Grumpkin is the same as the scalar field of BN256, the size of the scalar multiplication circuit for Grumpkin is about 25 times smaller than the non-native encoding of BN256.

To facilitate practical adoption we discuss integration aspects with Ethereum, provide performance and scalability benchmarks assessing the running time and the economic costs and savings associated with the deployment of our NI-DVRF and its on-chain verification on Ethereum which uses gas as a currency.

At a high glance, as summarised in Table 1, our NI-DVRF construction improves upon the existing threshold- and DVRF-based approaches, when considering key metrics regarding their performance and security, based on the recent analysis in [17, Table I]. Our NI-DVRF protocol achieves same levels of fault tolerance regarding honest majority, security and maximal damage as existing DRB solutions in [1, 15, 23, 29], while requiring a single communication round in the setup and randomness generation phases and providing optimal overall communication and verification complexities.

**Organisation.** In Section 2 we provide preliminaries and discuss the underlying building blocks and hardness assumptions. We present NI-DVRF definitions and their security requirements in Section 3. Our NI-DVRF construction and its correctness properties are described in Section 4. Section 5 focuses on its implementation, proposed optimisations for integration with Ethereum as well as

the experimental analysis of its performance and costs. In Section 6 we prove its security. We conclude in Section 7.

## 2 PRELIMINARIES

**Asymmetric Pairing Groups.** Let  $\mathbb{G}_1 = \langle g_1 \rangle$ ,  $\mathbb{G}_2 = \langle g_2 \rangle$  and  $\mathbb{G}_T$  be (cyclic) groups of prime order  $q$ . A map  $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$  a group  $\mathbb{G}_T$  is called a *bilinear* map, if it satisfies the following three properties:

- Bilinearity:  $e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$  for all  $x, y \in \mathbb{Z}_p$ .
- Non-Degenerate:  $e(g_1, g_2) \neq 1$ .
- Computable:  $e(g_1, g_2)$  can be efficiently computed.

*Definition 2.1.* The Strong Diffie-Hellman (SDH) assumption [7, 14] states that given  $(g, g^\alpha, g^\beta)$  and access to an oracle  $O_\beta(\cdot, \cdot)$  where  $O_\beta$  returns 1 on a query  $(U, X)$  such that  $U^\beta = X$  and 0 otherwise, it is hard to compute  $g^{\alpha\beta}$ .

*Definition 2.2.* The extended DDH assumption [8] holds if it is difficult to distinguish

$$(\mathbb{G}, q, g, g^{\alpha_1}, \dots, g^{\alpha_w}, g^\beta, y_1, \dots, y_w)$$

where  $y_i = g^{\alpha_i\beta}$  for all  $i \in \{1, \dots, w\}$  or randoms.

We observe that the extended DDH problem can be derived from the standard DDH problem: Consider  $w = 2$ . Given a DDH instance  $(\mathbb{G}, q, g, g^\alpha, g^\beta, y)$ , we can construct an extended DDH instance as follows:

$$\begin{aligned} s, r &\stackrel{\$}{\leftarrow} \mathbb{Z}_q \\ X &= g^s \cdot (g^\alpha)^r, Y = (g^\beta)^s \cdot y^r \\ \text{output } U &= (g^\alpha, X, g^\beta, y, Y). \end{aligned}$$

Assume  $y = g^\tau$  for some unknown  $\tau$ , we have  $X = g^{s+rr\alpha}$  and  $Y = g^{\beta s+rr\tau}$ , where  $s, r$  are uniformly distributed over  $\mathbb{Z}_q$ . When  $\tau = \alpha\beta$ , we have  $Y = X^\beta$  and  $U$  is indeed an extended DDH instance. When  $\tau \neq \alpha\beta$ , it is easy to verify that  $g^\alpha, X, g^\beta, y, Y$  are uniformly distributed and mutually independent which means  $U$  is an extended DDH instance. By repeating this process we can obtain extended DDH instances for general  $w$ .

*Definition 2.3 (co-CDH assumption).* Let  $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, q)$  be a bilinear groups. The co-CDH assumption states that given  $(g_1^\alpha, g_1^\beta, g_2^\alpha)$  with  $\alpha, \beta \stackrel{\$}{\leftarrow} \mathbb{Z}_q$ , it is hard to compute  $g_1^{\alpha\beta}$ .

*Definition 2.4 (XDH assumptions [12]).* Let  $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, q)$  be a bilinear groups. The XDH assumption states that DDH is hard in  $\mathbb{G}_1$ . The extended XDH assumption states that extended DDH is hard in  $\mathbb{G}_1$ .

**Shamir secret sharing [36].** A  $t$ -out-of- $n$  threshold secret sharing splits a secret  $s$  into  $n$  shares  $s_1, \dots, s_n$  such that any  $t$  shares are sufficient to recover the original value of  $s$ . The scheme involves selecting a random polynomial  $f(x)$  of degree  $t-1$  such that  $f(0) = s$  and computing the  $n$  shares as  $s_1 = f(1), \dots, s_n = f(n)$ . Any  $t$  points of  $f(x)$  can be combined to reconstruct  $f(x)$  using lagrange interpolation.

Scheme	Cryptographic Primitive	DKG setup			Randomness Generation			Fault tolerance (less than)	Security	Uniqueness	Withholding Immunity	Max damage
		No. rounds	Comm. complexity	Proof size	No. rounds	Comm. complexity	Verifier complexity					
Commit-reveal	Commitments	N/A			2	$O(n^2)$	$O(n)$	1	Unpredictability	×	×	Bias
Scrape [13]	PVSS	1	$O(n^2)$	N/A	2	$O(n^2)$	$O(n^2)$	$n/2$	INDI-Secrecy	×	✓	Bias
RandHound [37]	PVSS	1	$O(n^2)$	N/A	6	$O(c^2 n)$	$O(cn)$	$n/3$	Unpredictability/Unbiasability	×	×	Bias
Algorand [27]	VRP	1	$O(n^2)$	N/A	1	$O(n)$	$O(1)$	$n/3$	Unpredictability	✓	×	Bias
HERB[15]	Threshold ElGamal	6	$O(n^3)$	N/A	2	$O(n^2)$	$O(n)$	$n/3$	Unpredictability/Unbiasability	×	✓	Bias
Drand [1]	Threshold BLS	3	$O(n^3)$	N/A	1	$O(n^2)$	$O(1)$	$n/2$	Unpredictability/Unbiasability	✓	✓	Predict
DDH-DRB [23]	DDH-based DVRF	6	$O(n^3)$	N/A	1	$O(n^2)$	$O(n)$	$n/2$	Strong Pseudorandomness	✓	✓	Predict
GLOW-DRB [23]	Pairing-based DVRF	6	$O(n^3)$	N/A	1	$O(n^2)$	$O(1)$	$n/2$	Standard/Strong Pseudorandomness (relaxed) Unforgeability	✓	✓	Predict
Groth21 [29]	Threshold BLS, NIZK	1	$O(n^2 m)$	$O(n)$	1	$O(n^2)$	$O(1)$	$n/2$	Standard/Strong Pseudorandomness	✓	✓	Predict
This work	Threshold BLS, SNARK, DVRF	1	$O(n^3)$	$O(1)$	1	$O(n^2)$	$O(1)$	$n/2$	Standard/Strong Pseudorandomness	✓	✓	Predict

**Table 1: Comparison with existing DRB constructions.**  $n$  is the number of nodes participating in the execution of a DVRF-based DRB protocol. The number of rounds refers to the number of the coordinated rounds in a synchronous model. Each round involves all parties sending messages, waiting to receive messages, and then performing computations based on the received messages. Proof size is the size of the proof that each participant generates in a non-interactive DKG.  $m$  is the number of chunks in chunked encryption in Groth21 [29].  $c$  is the size of a shard in RandHound. Fault tolerance is the maximum number of corrupted nodes. Verifier complexity is the computational cost for a passive observer to verify the random output. Communication complexity is bitwise point-to-point communication amongst nodes. A withholding attack is an act where participants can influence the outcome by refraining from publishing certain information. Max damage refers to the maximum damage when  $n - 1$  rushing adversarial nodes cooperate.

*Definition 2.5 (Lagrange coefficients).* For a key reconstruction set  $\Delta$ , we define the *Lagrange basis polynomials*  $\lambda_{j,\Delta}(x) = \prod_{k \in \Delta \setminus \{j\}} \frac{x-k}{j-k} \in \mathbb{Z}_q[X]$  and the *Lagrange coefficients*  $\lambda_{i,j,\Delta} = \lambda_{j,\Delta}(i) \in \mathbb{Z}_q^*$ . For any polynomial  $f \in \mathbb{Z}_q[X]$  of degree at most  $|\Delta| - 1$  this entails  $\sum_{i \in \Delta} f(i) \lambda_{0,i,\Delta} = f(0)$ .

**Non-interactive zero knowledge proofs (NIZKs).** A NI proof system for a relation  $R$  produces a proof  $\pi \leftarrow \text{Prove}(\text{pk}, x, w)$  for a statement  $x$  and a witness  $w$ . The proof convinces a verifier that there exists a witness  $w$  such that  $(x, w) \in R$  and  $\text{Verify}(\text{vk}, x, \pi) = 1$ . The proving key  $\text{pk}$  and the verification key  $\text{vk}$  are created during the setup phase using  $(\text{pk}, \text{vk}) \leftarrow \text{Setup}(R)$ . The security properties of the proof system are:

- **Completeness:** given any true statement, an honest prover should be able to convince an honest verifier.
- **Zero knowledge:** the verifier does not learn any additional information about the witness beside the truth of the statement.
- **Knowledge soundness:** there is an extractor that can compute a witness whenever the adversary produces a valid argument.

The formal definitions are given in Appendix A.

### 3 NI-DVRF DEFINITIONS AND SECURITY REQUIREMENTS

This section defines non-interactive distributed verifiable random functions (NI-DVRFs), along with their standard and strong pseudorandomness properties, based on the formalization in [23]. Unlike the original definitions, the formalization in this section explicitly includes each member's public key and secret key. These keys are used in NI-DKG for encrypting and distributing secret shares among each other. In the security games for pseudorandomness, the challenger selects member public keys and member secret keys for honest members, while the adversary selects the keys for corrupted members.

*Definition 3.1 (NI-DVRF).* A  $t$ -out-of- $n$  NI-DVRF  $\mathcal{V} = (\text{KeyGen}, \text{DKG}, \text{PartialEval}, \text{Combine}, \text{Verify})$  consists of the following algorithms:

- $\text{KeyGen}(1^\lambda)$  takes as input a security parameter  $\lambda$  and creates a member secret key  $\text{msk}_i$  and a member public key  $\text{mpk}_i$  for a member  $i$ .
- $\text{NIDKG}(1^\lambda, t, n, \mathcal{M})$  is a non-interactive distributed *key generation* protocol that takes as input a security parameter  $1^\lambda$ , the threshold  $t$ , the total number of members  $n$ , the member secret-public key pairs  $\mathcal{M} = \{(\text{msk}_i, \text{mpk}_i)\}_{i=1}^n$ ; it outputs a set of qualified members  $\text{QUAL}$ , a global public key  $\text{gpk}$ , a list  $\mathcal{VK} = \{\text{vk}_1, \dots, \text{vk}_n\}$  of verification keys, and a list  $\mathcal{SK} = \{\text{sk}_1, \dots, \text{sk}_n\}$  of secret keys where each secret key is only known to the corresponding member.
- $\text{PartialEval}(x, \text{sk}_i, \text{vk}_i)$  takes as input a plaintext  $x$ , a secret key  $\text{sk}_i$  and a verification key  $\text{vk}_i$ , and outputs  $\sigma_x^i = (i, v_i, \pi_i)$ , where  $v_i$  is the  $i$ -th evaluation share and  $\pi_i$  is a proof of correct partial evaluation.
- $\text{PartialVerify}(x, \text{vk}_i, \sigma_x^i)$  takes as input a plaintext  $x$ , a verification key  $\text{vk}_i$  and a partial evaluation  $\sigma_x^i$ , and outputs 0/1.
- $\text{Combine}(\mathcal{VK}, x, E)$  takes as input the verification keys  $\mathcal{VK}$ , a plaintext  $x$ , and a set  $E = \{\sigma_x^{i_1}, \dots, \sigma_x^{i_{|E|}}\}$  of partial evaluations from  $|E| \geq t$  distinct members, and outputs either a pair  $(v, \pi)$  of pseudorandom value  $v$  and correctness proof  $\pi$ , or  $\perp$ .
- $\text{Verify}(\text{gpk}, x, v, \pi)$  takes as input the global public key  $\text{gpk}$ , a plaintext  $x$ , a pseudorandom value  $v$  and a proof  $\pi$ , and outputs 0/1.

A NI-DVRF is *correct* if it satisfies the following requirements of robustness and uniqueness:

- **Robustness:** in the presence of the adversary's inputs to the Combine algorithm, if Combine does not return  $\perp$  then its output must pass the verification test. Robustness guarantees the availability of computing the random function value on any plaintext in the presence of an active adversary.

- *Uniqueness*: for any plaintext  $x$ , a unique value  $v$  passes the verification test. In other words, it is impossible for any adversary to compute two different values  $v, v'$  for  $x$  such that both values pass the verification test, even when the secret keys of the honest nodes are leaked.

Their formal definitions can be found in Appendix B.

Standard pseudorandomness requires that the pseudorandom value is indistinguishable from a uniform random. Strong pseudorandomness is similar to standard pseudorandomness, except that the adversary is allowed to query the partial evaluation oracle on the challenge plaintext up to  $t - |C| - 1$  times where  $C$  is a collection of corrupted members. In the following definitions, we restrict the number of corrupted members to be  $|C| < t \leq n - |C|$ , which is equivalent to  $|C| < \min\{t, n/2\}$ . Therefore, the security of our NI-DVRF protocol is based on the assumption of an honest majority.

*Definition 3.2 (Standard Pseudorandomness)*. A NI-DVRF protocol  $\mathcal{V} = (\text{KeyGen}, \text{NIDKG}, \text{PartialEval}, \text{PartialVerify}, \text{Combine}, \text{Verify})$  is *pseudorandom* if for all PPT adversaries  $\mathcal{A}$ , there exists a negligible function  $\text{negl}(\cdot)$  such that

$$\left| \Pr[\text{PRand}_{\mathcal{V}, \mathcal{A}}(1^\lambda, 0) = 1] - \Pr[\text{PRand}_{\mathcal{V}, \mathcal{A}}(1^\lambda, 1) = 1] \right| \leq \text{negl}(\lambda)$$

where  $\text{PRand}_{\mathcal{V}, \mathcal{A}}(1^\lambda, b)$  is the experiment defined below:

*Corruption* On input a list of members  $P = \{P_1, \dots, P_n\}$  and threshold  $1 \leq t \leq n$ , an adversary  $\mathcal{A}$  selects a collection  $C$  ( $|C| < t \leq n - |C|$ ) of members to corrupt and gives  $C$  to the challenger. Adversary  $\mathcal{A}$  acts on behalf of corrupted members, while the challenger acts on behalf of the remaining members, which behave honestly (namely they follow the protocol specification). The challenger chooses member public keys for honest members and sends them to the adversary. The adversary chooses member public keys for corrupted members and sends them to the challenger.

*Initialization* Challenger and adversary runs the NI-DKG protocol  $\text{NIDKG}(1^\lambda, t, n)$ . After this phase, the protocol establishes a qualified set of members  $\text{QUAL}$ . Every (honest) member  $P_j \in \text{QUAL} \setminus C$  obtains a key pair  $(sk_j, vk_j)$ . In contrast, (corrupted) members  $P_j \in C$  end up with key pairs  $(sk_j, vk_j)$  in which one of keys may be undefined (i.e. either  $sk_j = \perp$  or  $vk_j = \perp$ ). At the end of this phase, the global public key  $\text{gpk}$  and the verification keys  $\{vk_i\}_{i \in \text{QUAL}}$  are known by both the challenger and the adversary.

*Pre-Challenge Evaluation* In response to  $\mathcal{A}$ 's evaluation query  $(\text{Eval}, x, i)$  for some honest member  $P_i \in \text{QUAL} \setminus C$  and plaintext  $x$ , the challenger returns  $\sigma_x^i \leftarrow \text{PartialEval}(sk_i, vk_i, x)$ . In any other case, the challenger returns  $\perp$ .

*Challenge* The challenger receives from the adversary  $\mathcal{A}$  a set  $U \subseteq \text{QUAL}$  with  $|U| \geq t$ , a plaintext  $x^*$  such that  $(\text{Eval}, x^*, *)$  has never been queried, and a set of partial evaluation shares  $\{\sigma_{x^*}^i\}_{P_i \in U \cap C}$ . Let  $\sigma_{x^*}^j \leftarrow \text{PartialEval}(sk_j, vk_j, x^*)$  for  $P_j \in U \setminus C$  and  $(v^*, \pi^*) \leftarrow \text{Combine}(\text{pk}, \text{vk}, x^*, \{\sigma_{x^*}^j\}_{P_j \in U})$ . If  $v^* = \perp$  the experiment output  $\perp$ . Otherwise, if  $b = 0$  the adversary receives  $v^*$ ; if  $b = 1$  the adversary receives a uniform random.

*Post-Challenge Evaluation* In response to  $\mathcal{A}$ 's query  $(\text{Eval}, x, i)$  with  $x \neq x^*$  for some member  $P_i \in \text{QUAL} \setminus C$  and plaintext  $x \in \mathcal{D}$ , the challenger returns  $\sigma_x^i \leftarrow \text{PartialEval}(sk_i, vk_i, x)$ . In any other case, the challenger returns  $\perp$ .

*Guess* Finally,  $\mathcal{A}$  returns its guess  $b' \in \{0, 1\}$  which is then output by the game.

*Definition 3.3 (Strong Pseudorandomness)*. Strong pseudorandomness is defined exactly the same as standard pseudorandomness except that the adversary is allowed to query  $(\text{Eval}, x^*, *)$  on the challenge plaintext  $x^*$  for up to  $t - |C| - 1$  times in total before and after the challenge query.

## 4 OUR NI-DVRF CONSTRUCTION

We present a construction for NI-DVRF using threshold secret sharing, SNARKs, and BLS signatures. This protocol runs among a group of  $n$  members. Each member has a pair of member secret key and member public key that will be used to encrypt (decrypt) shares to (from) other members in a SNARK-based non-interactive distributed key generation protocol. At the end of the protocol, a global public key, a list of secret keys and a list of verification keys are established. After this, members can use their keys to repeatedly generate pseudo-random values for multiple epochs.

Let  $(\mathbb{G}, g, p)$  be a cyclic group, and  $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, q)$  be a bilinear groups. Let  $H_1 : \mathbb{G} \mapsto \mathbb{Z}_q, H_2 : \{0, 1\}^* \mapsto \mathbb{G}_1, H_3 : \{0, 1\}^* \mapsto \mathbb{Z}_q$  and  $H_4 : \mathbb{G}_1 \mapsto \{0, 1\}^*$  be hash functions. Let  $\mathcal{V}^{\text{NI-DVRF}} = (\text{KeyGen}, \text{NIDKG}, \text{PartialEval}, \text{PartialVerify}, \text{Combine}, \text{Verify})$  be a NI-DVRF constructed as follows:

**KeyGen** $(1^\lambda)$ : select  $\text{msk}_i \xleftarrow{\$} \mathbb{Z}_p$  uniformly at random and compute  $\text{mpk}_i = g^{\text{msk}_i}$ . Output  $(\text{msk}_i, \text{mpk}_i)$ .

**NIDKG** $(1^\lambda, t, n, \mathcal{M})$ : this protocol establishes a global public key  $\text{gpk}$ , each member  $i$  obtains a DKG secret key  $sk_i$  and the corresponding (public) verification key  $vk_i$ .

- (1) Each member  $i$  chooses a random polynomial and evaluates it at  $n$  points:

$$f_i(x) = a_{i,0} + a_{i,1}x + \dots + a_{i,t-1}x^{t-1}$$

$$s_{i,1} = f_i(1), s_{i,2} = f_i(2), \dots, s_{i,n} = f_i(n)$$

$s_{i,1}, \dots, s_{i,n}$  are the secret shares that are capable of recovering the polynomial  $f_i(x)$ . Member  $i$  computes public parameters for  $f_i(x)$ :

$$pp_i = (g_1^{s_{i,1}}, g_1^{s_{i,2}}, \dots, g_1^{s_{i,n}}, g_1^{a_{i,0}}, g_2^{a_{i,0}})$$

Member  $i$  gives the secret shares to corresponding members by encrypting  $s_{i,j}$  using member  $j$ 's public key:

$$\text{cipher}_i = (h, c_{i,1}, c_{i,2}, \dots, c_{i,n})$$

where  $h = g^r, c_{i,j} = H_1(\text{mpk}_j^r) + s_{i,j}$ . Member  $i$  publishes  $(pp_i, \text{cipher}_i)$  together with a SNARK proof to show its data is formed correctly (more details in Section 5). The encryption  $\text{cipher}_i$  is generated on group  $\mathbb{G}$  which does not have to be the same as the pairing group  $\mathbb{G}_1$ . In the implementation, we can choose  $\mathbb{G}$  to be a group that is natively compatible with SNARK implementation which can significantly improve efficiency.

- (2) Upon receiving  $\{(pp_j, cipher_j)\}_j$  from other members, member  $i$  verifies the validity of the SNARK proofs. Since some members may not participate or provide invalid data, the final qualified set is a subset  $QUAL \subseteq \{1, 2, \dots, n\}$ . Member  $i$  decrypts the cipher  $c_{j,i}$  in  $cipher_j$  for  $j \in QUAL$  to recover the secret shares  $\{s_{j,i}\}_{j \in QUAL}$ :  $s_{j,i} \leftarrow c_{j,i} - H_1(h^{msk_i})$ . From these shares, member  $i$  derives its final DKG secret key  $sk_i = \sum_{j \in QUAL} s_{j,i}$  and verification key  $vk_i = g_1^{sk_i}$ .
- (3) The public does not have the decryption keys to access any of the secret shares, but can still verify the SNARK proofs to determine the qualified set  $QUAL$  and valid  $\{(pp_j, cipher_j)\}_{j \in QUAL}$ . The public can derive the public parameters as follows:

- Global public key is set to  $gpk = \prod_{j \in QUAL} g_2^{a_{j,0}}$ .
- Each member  $i$ 's DKG verification key can be computed as  $vk_i = \prod_{j \in QUAL} g_1^{s_{j,i}}$ .

**PartialEval**( $x, sk_i, vk_i$ ): compute  $v_i = H_2(x)^{sk_i}$ . As  $v_i$  and  $vk_i$  are both on  $\mathbb{G}_1$ , a Schnorr-style NIZK proof  $\pi_i = (c, z)$  is generated to show  $v_i$  is formed correctly, i.e.,  $v_i$  and  $vk_i$  have the same discrete logarithm:  $e \xleftarrow{\$} \mathbb{Z}_p, R_1 = g_1^e, R_2 = H_2(x)^e, c = H_3(g_1, H_2(x), vk_i, v_i, R_1, R_2), z = sk_i * c + e$ . Output the partial evaluation  $\sigma_x^i = (i, v_i, \pi_i)$ .

**PartialVerify**( $x, vk_i, \sigma_x^i$ ): parse  $\sigma_x^i = (i, v_i, \pi_i)$ , verify if  $\pi_i = (c, z)$  is valid as follows:  $\tilde{R}_1 = g_1^z / vk_i^c, \tilde{R}_2 = H_2(x)^z / v_i^c, \tilde{c} = H_3(g_1, H_2(x), vk_i, v_i, \tilde{R}_1, \tilde{R}_2)$  and output  $c \stackrel{?}{=} \tilde{c}$ .

**Combine**( $\mathcal{VK}, x, E$ ): parse the list  $E = \{\sigma_x^{j_1}, \dots, \sigma_x^{j_{|E|}}\}$  of  $|E| \geq t$  partial evaluations from  $|E|$  different members, and obtain verification keys  $vk_{j_1}, \dots, vk_{j_{|E|}}$ . Next,

- (1) Identify an index subset  $I = \{i_1, \dots, i_t\}$  such that for every  $i \in I$  it holds that  $\text{PartialVerify}(x, vk_i, \sigma_x^i) = 1$ . If no such subset exists, outputs  $\perp$ .
- (2) Set  $\pi \leftarrow \prod_{j \in I} v_j^{\lambda_{0,j,I}}$  and  $v = H_4(\pi)$ .
- (3) Output  $(v, \pi)$ .

**Verify**( $gpk, x, v, \pi$ ): output 1 if the relation holds:  $e(\pi, g_2) = e(H_2(x), gpk)$  and  $v = H_4(\pi)$ . Otherwise output 0.

**Correctness properties.** It is easy to verify that the robustness and uniqueness holds for our construction due to the fact that only one  $\pi = H_2(x)^{gsk}$  is valid and can pass the Verify algorithm for each input  $x$ , where  $gsk$  is the implicit secret key underlying the global public key  $gpk$ .

**THEOREM 4.1.**  $\mathcal{V}^{\text{NI-DVRF}}$  is robust and unique.

**PROOF (SKETCH).** It is easy to see that

- The global public parameters ( $gpk, \mathcal{VK}$ ) and the secret keys  $SK$  of the honest members are correctly formed at the end of NIDKG due to the extractability of SNARK proofs.
- Combining partial evaluations from any subset  $\Delta \subseteq QUAL$  with  $|\Delta| \geq t$  gives the same value  $H_1(x)^{gsk}$  due to the

following equality:

$$\begin{aligned} \sum_{j \in \Delta} sk_j \lambda_{0,j,\Delta} &= \sum_{j \in \Delta} \lambda_{0,j,\Delta} \left( \sum_{i \in QUAL} s_{i,j} \right) \\ &= \sum_{i \in QUAL} \left( \sum_{j \in \Delta} \lambda_{0,j,\Delta} \cdot s_{i,j} \right) \\ &= \sum_{i \in QUAL} \left( \sum_{j \in \Delta} \lambda_{0,j,\Delta} \cdot f_i(j) \right) = \sum_{i \in QUAL} a_{i,0} = gsk \\ \prod_{j \in \Delta} (H_1(x)^{sk_j})^{\lambda_{0,j,\Delta}} &= H_1(x)^{\left( \sum_{j \in \Delta} \lambda_{0,j,\Delta} \cdot sk_j \right)} = H_1(x)^{gsk} \end{aligned}$$

holds for every subset  $\Delta \subseteq QUAL$  with  $|\Delta| \geq t$ .

Robustness holds because Combine function verifies the NIZK proofs to guarantee the validity of the partial evaluations. Thus robustness can be proven using the extractability of the NIZKs. When the challenger outputs 1, it means  $v^* \neq \perp$  and  $\text{Verify}(gpk, x^*, v^*, \pi^*) = 0$ . W.l.o.g., assume  $|U| = t$ . Because of the above equations, we can derive that there exists  $i \in U$  such that  $v_i \neq H_1(x^*)^{sk_i}, vk_i = g_1^{sk_i}$  and  $\pi_i$  a verified NIZK proof. We consider two cases of  $i$ . If  $i \in U \setminus C$ , this is impossible since  $v_i$  is computed correctly by the challenger. If  $i \in U \cap C$ , we can use the PPT extractor of the NIZKs to derive a  $k$  such that  $v_i = H_1(x^*)^k$  and  $vk_i = g_1^k$  which contradicts the hypothesis.

The uniqueness follows from the fact that  $v = H_2(\pi), v' = H_2(\pi')$  and if  $e(\pi, g_2) = e(H_1(x), gpk) = e(\pi', g_2)$  then  $\pi = \pi' = H_1(x)^{gsk}$ .  $\square$

## 5 ETHEREUM-COMPATIBLE IMPLEMENTATION

Our NI-DVRF protocol has been implemented<sup>1</sup> and is currently being investigated for the adoption on the Boba Network to power its DRB service zkRand.<sup>2</sup> Our design and implementation are customised to be compatible with Ethereum and Ethereum-like chains to facilitate the fast adoption in crypto industry. The main protocol is implemented in Rust. Additionally, Solidity contracts were developed to enable on-chain verification on Ethereum. The evaluation of performance has been performed on an AWS instance r6i.8xlarge, which has 32 CPUs and 256GB of memory.

**NI-DKG.** We use Halo2 with KZG commitment [4] on the BN256 curve to generate the SNARK proof for our NI-DKG protocol. Halo2 supports Plonk-ish style circuits. Our NI-DKG circuit checks that the following computations are performed correctly:

- Secret shares  $s_{i,1}, \dots, s_{i,n}$  are evaluated consistently from the same coefficients  $a_{i,0}, \dots, a_{i,t-1}$ .
- Public parameters  $pp_i = (g_1^{s_{i,1}}, g_1^{s_{i,2}}, \dots, g_1^{s_{i,n}}, g_1^{a_{i,0}}, g_2^{a_{i,0}})$  are generated correctly using  $s_{i,1}, \dots, s_{i,n}, a_{i,0}$ .
- The secret shares are encrypted correctly with respect to the given member public keys. This ensures that decryption can be performed by the corresponding members.

In the circuit, the public parameters  $pp_i$  are computed using non-native encoding of the BN256 curve from halo2wrong [5]. As our

<sup>1</sup>Implementation can be found at <https://github.com/bobanetwork/zkdvrf>

<sup>2</sup><https://bobanetwork.medium.com/zkrand-demo-bc40785fdb>

protocol only requires a fixed generator  $g_1$  to create public shares, we have developed a windowed scalar multiplication chip for fixed point generator, which reduces 70% of gates.

We have developed the  $\mathbb{G}_2$ -chip for performing scalar multiplications in  $\mathbb{G}_2$ . This chip allows us to verify that  $g_2^{a_i,0}$  is computed correctly within the circuit. The computation of  $\mathbb{G}_2$  operations is around 3 times larger than  $\mathbb{G}_1$  operations, which is why we choose to compute most of the public parameters in  $\mathbb{G}_1$ . We only need to compute one scalar multiplication in  $\mathbb{G}_2$ , which adds a constant computation overhead. Consequently, the operations on  $\mathbb{G}_2$  increase the circuit size for small values of  $n$ , but this increase can be disregarded when  $n$  is large, as shown in Table 2 (more explanations below). Alternatively, the correspondence between  $g_1^{a_i,0}$  and  $g_2^{a_i,0}$  can be checked using pairing equations outside the SNARK proof. However, this will incur additional gas cost when verifying the proof on-chain.

The encryption  $cipher_i$  of shares are performed on Grumpkin curve (instead of BN256) for which we have developed ecc-chip to generate circuits. The scalar multiplication circuit for Grumpkin is currently a double-and-add method with optimisations customised for Halo2wrong maingate to reduce the number of gates. Since the base field of Grumpkin is the same as the scalar field of BN256, the size of the scalar multiplication circuit for Grumpkin is about 25 times smaller than the non-native encoding of BN256. The hash function used in  $cipher_i$  is Poseidon hash.

Our performance evaluation results are summarised in Table 2: Table 2a evaluates NI-DKG circuits without our  $\mathbb{G}_2$  chip whereas Table 2b utilises the  $\mathbb{G}_2$  chip. In Halo2 proof systems, circuits need to be padded to the nearest power of 2. Given a degree  $k$ , the maximum number of gates the circuit can have is  $2^k$ . In Tables 2a and 2b, the total number of members  $n$  is chosen to be the maximum number that can be supported under a specific circuit degree and the threshold  $t$  is chosen to be the majority of  $n$ . The values of  $(t, n)$  in Table 2b are lower than the ones in Table 2a due to the computation for  $\mathbb{G}_2$  operations. However this impact becomes less significant as  $n$  increases because only one  $\mathbb{G}_2$  scalar multiplication is required. The proof size remains constant, while the verification time increases with  $n$  due to the size of public instances (i.e., the size of  $pp_i$  and  $cipher_i$ ). In the SNARK verification, there is polynomial evaluation for processing the public instances. The verification time can be reduced to be constant by hashing the public inputs, but choosing the appropriate hash function is challenging. Poseidon hash is SNARK-friendly and imposes minimal computation overhead on the prover. However, computing the Poseidon hash in Solidity contract is expensive because there are no precompiles available for it. Keccak and Sha256 are cheap to compute in Solidity but they significantly increases the proving time and memory usage. We leave further optimizations for verification time and cost as future work. For example, we may explore the use of recursive SNARKs to reduce on-chain verification costs. As shown in Table 2, generating a SNARK proof for approximately 40 members requires a peak memory usage of 16GB, which is considered moderate and can be easily handled by off-the-shelf computers. In practice, 40 nodes should be sufficient for most applications. For example, Drand [1] began with a threshold of 6-out-10 in 2019 and currently operates 23 nodes with a threshold of 12 to create verifiable randomness.

**Randomness generation.** Once the NI-DKG setup is complete, members can use their DKG secret keys to generate pseudo-random values for multiple epochs. The computation required for generating these values is lightweight and can be easily performed on any commodity laptop. The evaluation results are given in Table 3. The performance of a single partial evaluation, its verification, and the verification of the final pseudorandom value are independent of the values of  $(t, n)$ . The evaluation for combine do not include the time for validating partial evaluations. It is worth pointing out that each member's verification key  $vk_i$  is created on  $\mathbb{G}_1$  as well as the partial evaluation  $v_i$ . Consequently, the validity of  $v_i$  is proven using a Schnorr-style NIZK proof instead of pairing equation. This results in a 1.6 times faster verification, and saves 47% - 62% gas costs for on-chain verification as shown in Table 4.

**Ethereum integration considerations.** Ethereum has precompiles for BN256 curve, thus we developed solidity contracts to verify the SNARK proofs generated in NI-DKG, check the validity of partial evaluations, and ensure the correctness of the final pseudo-random value. The gas costs for these verification algorithms are shown in Table 4. The gas cost for verifying the SNARK proofs currently starts at 700k due to several design choices made in Halo2wrong, such as 5-width advice columns. It may be possible to reduce the gas costs by refactoring Halo2wrong to use fewer advice columns. In addition, we shall discuss some general strategy for saving gas costs later. At the time of experiments, 700k gas was approximately \$26 on Ethereum mainnet. Transaction fees on Ethereum-equivalent or EVM-compatible L2 networks are much cheaper, for example,  $\sim 10$  times cheaper on Optimism and zkSync, and  $\sim 100$  times cheaper on Boba Network, which makes it cost effective to deploy our protocol. On the other hand, NI-DKG is the initialisation process that only needs to be executed once throughout the protocol and does not have to occur frequently.

Ethereum precompiles on BN256 do not support some of the necessary operations required by our DVRF. Below we suggest some workarounds:

- Since there is no precompiles for addition on  $\mathbb{G}_2$ , the value of  $gpk$  can be computed off-chain and then be verified on-chain by computing  $\omega = \prod_{j \in Q} g_1^{a_j,0}$  and checking if  $e(\omega, g_2) = e(g_1, gpk)$ .
- Ethereum does not have precompiles for  $hash\_to\_curve$ , i.e.,  $H_2(x)$ , so we have implemented the function in Solidity contract and the gas cost is ranging between 55k-70k.  $hash\_to\_curve$  is computed during the verification of partial evaluations and the final pseudorandom value. As shown in Table 4, verifying a partial evaluation costs 101392 gas and verifying a final pseudorandom value costs 193693. However, for each  $x$ , the value of  $H_2(x)$  only needs to be computed once and can be stored in the contract. With this precomputation, the gas cost be reduced to 55k for verifying the partial evaluation and 147k for verifying the final pseudorandom value.

The ability to verify data and proofs on-chain is crucial for resolving disputes in a decentralized application environment. For instance, verifying partial evaluations on chain helps determine which members should be rewarded and which members should be

circuit degree	$(t, n)$	snark-prove (s)	snark-verify (ms)	snark proof size (Bytes)	peak memory usage (GB)
18	(5,9)	19.908	5.1817	3488	4.6
19	(11,21)	37.616	5.5494		8.8
20	(22, 43)	74.689	6.2203		16.6
21	(45, 88)	147.650	7.5934		32.6
22	(89, 176)	295.792	10.270		64.4

(a) NI-DKG circuits without  $\mathbb{G}_2$  chip

circuit degree	$(t, n)$	snark-prove (s)	snark-verify (ms)	snark proof size (Bytes)	peak memory usage (GB)
18	(3,5)	20.758	5.0838	3488	4.8
19	(9,16)	38.055	5.4085		8.8
20	(20, 38)	74.738	6.0364		16.5
21	(42, 83)	148.438	7.3965		32.5
22	(86, 171)	294.286	10.139		64.4

(b) NI-DKG circuits with  $\mathbb{G}_2$  chip

**Table 2: NI-DKG performance evaluation.** This table evaluates the time it takes to generate and verify SNARK proof for the NI-DKG circuit. Circuit degree determines the maximum number of gates.  $n$  is chosen to be the maximum number that can be supported under a specific degree and  $t$  is the majority of  $n$ . The memory usage represents the peak memory used for generating the proof. The proving time and memory usage are linear in the size of the circuit. The proof size is constant. The verification time increases due to the size of public instances (i.e.,  $pp_i$  and  $cipher_i$ ).

$(t, n)$	create-partial-eval (ms)	verify-partial-eval (ms)	combine (ms)	verify-pseudo-random (ms)
(3,5)	0.856	1.0262	0.650	1.6194
(9,16)			1.9135	
(20, 38)			4.2424	
(42, 83)			8.9423	
(86, 171)			18.517	

**Table 3: Randomness generation.** Timing results for creating and verifying a single partial evaluation, combining  $t$  partial evaluations and verifying the final pseudorandom value. The combining algorithm is linear in  $t$  and its evaluation does not include the time for validating partial evaluations. Verification of the final pseudorandom value takes constant time.

punished, providing economic incentives for members to participate and behave accordingly. However, it is not necessary to verify every piece of data as it is sent on-chain. To save on gas costs, we can employ a **lazy verification** strategy: the data owner locks away a deposit for a specific period, and anyone can run the verification algorithm to challenge the correctness of the data within this period. If the verification fails, the defender (i.e., the data owner) will have their deposit slashed and the challenger receives rewards. On the other hand, if the verification is successful, the challenger will pay for the verification cost. This approach significantly reduces the actual operational costs for the protocol.

**Scalability.** In blockchain applications, a moderate number of nodes, usually between 10 to 30, is often sufficient. For example, Drand currently operates 23 nodes to generate random numbers. If there are a large number of nodes, such as 160, a more practical and efficient strategy is to divide these nodes into smaller committees. These could be, for instance, 10 committees each with 16 nodes.

Each committee would then run a NI-DKG with peak memory usage of less than 10GB which can be easily computed on a laptop. The different committees could then rotate for generating random numbers based on the previous beacon outputs. This is a more feasible approach than directly running a NI-DKG with 160 nodes, and then summoning 80 nodes to generate random numbers for each round.

**Further discussion.** The most relevant related work for our contributions is [1, 23, 29], however comparing their experimental results for DKG is not meaningful for the following reasons: [1, 23] uses an interactive DKG with multiple rounds of communications among all the participants. As for Groth21[29], this protocol is implemented using bls12-381 curve instead of bn254. However, bls12-381 is slower than bn254 and is *not* supported on Ethereum. Moreover, verification in [29] requires exponentiations on  $G_2$  which are not supported on Ethereum either. Since our construction eliminated the need for chunked encryption used in Groth21, we removed the factor of  $m$  as shown in Table 1.

circuit degree	$(t, n)$	verify snark-proof	verify partial-eval	verify (fast) partial-eval	verify pseudo-random	verify (fast) pseudo-random
18	(3,5)	726115				
19	(9,16)	808112				
20	(20, 38)	972917	101392	55098	193693	147468
21	(42, 83)	1312117				
22	(86, 171)	1985415				

Table 4: Cost for on-chain verification on Ethereum in gas currency.

## 6 SECURITY ANALYSIS

In this section, we formally prove that our  $\mathcal{V}^{\text{NI-DVRF}}$  construction achieves both standard pseudorandomness and strong pseudorandomness in the random oracle model. Pseudorandomness requires that the pseudorandom value is indistinguishable from a uniformly random value from the attacker's point of view. Standard pseudorandomness prohibits the adversary to query the challenge plaintext, while the strong pseudorandomness allows the queries of the challenge plaintext for up to  $t - 1$  times.

**THEOREM 6.1.**  $\mathcal{V}^{\text{NI-DVRF}}$  achieves standard pseudorandomness under the co-CDH assumption and SDH assumption in the random oracle model.

**PROOF.** We first construct  $\text{Hyb}_{\mathcal{A}}^{\text{snark}}(b)$  to simulate SNARK proof in NI-DKG and the original standard pseudorandomness game is indistinguishable with  $\text{Hyb}_{\mathcal{A}}^{\text{snark}}(b)$  because of the zero knowledge property of SNARK.

We construct  $\text{Hyb}_{\mathcal{A}}^{\text{enc}}(b)$  in the exact same way as  $\text{Hyb}_{\mathcal{A}}^{\text{snark}}(b)$  except all the shares  $s_{i,j}$  with  $j \in [m+1, n]$  encrypted in  $\text{cipher}_i$  with  $i \in [m+1, n]$  are replaced with uniform randoms. It is easy to see that  $\text{Hyb}_{\mathcal{A}}^{\text{enc}}(b)$  and  $\text{Hyb}_{\mathcal{A}}^{\text{snark}}(b)$  are indistinguishable under SDH assumption. We shall sketch the proof here. W.l.o.g., assume adversary chooses to corrupt members  $C = \{1, \dots, m\}$ . Given an SDH problem  $(G, q, g, g^\alpha, g^\beta)$  with oracle  $O_\beta$ ,

- Challenger chooses randoms  $\rho_{m+1}, \dots, \rho_n \xleftarrow{\$} \mathbb{Z}_q$  and sets up the public keys for honest members  $j \in [m+1, n]$  as  $\text{mpk}_j = g^{\alpha \rho_j}$
- Challenger selects randoms  $r_{m+1}, \dots, r_n, z_{m+1,1}, \dots, z_{m+1,n}, \dots, z_{n,1}, \dots, z_{n,n} \xleftarrow{\$} \mathbb{Z}_q$ . The encryption  $\text{cipher}_i$  of shares from the honest members  $i \in [m+1, n]$  are computed as

$$\text{cipher}_i = (g^{\beta r_i}, z_{i,1} + s_{i,1}, \dots, z_{i,n} + s_{i,n})$$

Let  $U_{i,j} = \text{mpk}_j^{r_i}$  with  $i \in [m+1, n]$  and  $j \in [1, n]$ . On query  $x$ , the oracle  $H_1$  is simulated as follows: Define a list  $\mathcal{L}_{H_1} = \emptyset$ .

- If there exists  $(x, c) \in \mathcal{L}_{H_1}$ , then return  $c$
- Otherwise  $x$  has not been queried before,
  - \* If  $O_\beta(U_{i,j}, x) = 1$  for some  $i \in [m+1, n]$  and  $j \in [1, n]$ , then update the list  $\mathcal{L}_{H_1} = \mathcal{L}_{H_1} \cup \{(x, z_{i,j})\}$  and return  $z_{i,j}$ . If  $m+1 \leq j \leq n$ , then output  $x^{r_i^{-1} \rho_j^{-1}}$  as solution to SDH problem. We call this event  $E$ .
  - \* Else choose random  $z$ , update the list  $\mathcal{L}_{H_1} = \mathcal{L}_{H_1} \cup \{(x, z)\}$  and return  $z$ .

If  $E$  does not happen,  $z_{i,j}$  with  $i \in [m+1, n], j \in [m+1, n]$  are uniform randoms from the adversary's point of view. Therefore adversary sees the same distribution whether the shares  $s_{i,j}$  with  $i \in [m+1, n], j \in [m+1, n]$  are correct shares (for  $\text{Hyb}_{\mathcal{A}}^{\text{snark}}(b)$ ) or randoms (for  $\text{Hyb}_{\mathcal{A}}^{\text{enc}}(b)$ ). If  $E$  happens, it solves the SDH problem. That is  $|\Pr[\text{Hyb}_{\mathcal{A}}^{\text{snark}}(b) = 1] - \Pr[\text{Hyb}_{\mathcal{A}}^{\text{enc}}(b) = 1]| \leq \text{Adv}^{\text{sdh}}$ .

Next, we construct  $\text{Hyb}_{\mathcal{A}}^{\text{sim}}(b)$  in the exact same way as  $\text{Hyb}_{\mathcal{A}}^{\text{enc}}(b)$  except all the zero-knowledge proofs for partial evaluations generated from the honest members are replaced with simulated proofs.  $\text{Hyb}_{\mathcal{A}}^{\text{sim}}(b)$  and  $\text{Hyb}_{\mathcal{A}}^{\text{enc}}(b)$  are indistinguishable because of the zero knowledge property.

Below we shall prove  $\text{Hyb}_{\mathcal{A}}^{\text{sim}}(0)$  and  $\text{Hyb}_{\mathcal{A}}^{\text{sim}}(1)$  are indistinguishable under co-CDH assumption. Suppose there exists an adversary  $\mathcal{A}$  that distinguishes  $\text{Hyb}_{\mathcal{A}}^{\text{sim}}(0)$  and  $\text{Hyb}_{\mathcal{A}}^{\text{sim}}(1)$ , then we can construct an adversary  $\mathcal{B}$  breaks the co-CDH assumption using  $\mathcal{A}$  as a subroutine.

Given a co-CDH problem  $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2, g_1^\alpha, g_1^\beta, g_2^\alpha)$  with  $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2, \alpha, \beta \xleftarrow{\$} \mathbb{Z}_q$ .  $\mathcal{B}$ 's goal is to output  $g_1^{\alpha\beta}$ :

- (1) Give the public parameters  $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2)$  to  $\mathcal{A}$ .
- (2)  $\mathcal{A}$  chooses a set  $C$  of members with  $|C| < t \leq n - |C|$  to corrupt. W.l.o.g., assume  $C = \{1, 2, \dots, m\}$ .  $\mathcal{A}$  gives  $C$  to  $\mathcal{B}$ .  $\mathcal{B}$  chooses public keys  $\text{mpk}_i$  for honest members  $i \in [m+1, n]$  and gives  $\{\text{mpk}_i\}_{m+1 \leq i \leq n}$  to  $\mathcal{A}$ .  $\mathcal{A}$  chooses public keys  $\text{mpk}_i$  for  $i \in C$  and sends  $\{\text{mpk}_i\}_{i \in C}$  to  $\mathcal{B}$ .
- (3) The random oracle  $H_2$  is answered as follows. Initialise  $\mathcal{L}_{H_2} = \emptyset$ . Let  $q_{H_2}$  be the total number of distinct random oracle queries asked in this game. Choose an index  $\eta^* \xleftarrow{\$} [q_{H_2}]$  uniformly at random. On query  $x$ ,
  - If there exists a tuple  $(x, r, h) \in \mathcal{L}_{H_2}$ , output  $h$ .
  - Otherwise,
    - if this is the  $\eta^*$ -th distinct call, set  $r = \perp$  and  $h = g_1^\beta$  where  $g_1^\beta$  is from the co-CDH problem.
    - else choose a random  $r \xleftarrow{\$} \mathbb{Z}_q$  and set  $h = g_1^r$ .
    - Update  $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup \{(x, r, h)\}$  and output  $h$ .

Give random oracle access to  $\mathcal{A}$ .
- (4) The random oracle  $H_4$  is programmed as follows: Define a list  $\mathcal{L}_{H_4} = \emptyset$ . For a query on  $y$ ,
  - Before NI-DKG is completed, the oracle is answered in the standard way,
    - If there exists  $(y, c) \in \mathcal{L}_{H_4}$ , then return  $c$
    - Otherwise, choose a random  $c \xleftarrow{\$} \mathbb{Z}_q$ , update the list  $\mathcal{L}_{H_4} = \mathcal{L}_{H_4} \cup \{(y, c)\}$ , then return  $c$

- After NI-DKG is completed, the global public key  $\text{gpk}$  has been established,
  - If  $e(g_1^\beta, \text{gpk}) = e(y, g_2)$  and there exists  $(\perp, c) \in \mathcal{L}_{H_4}$  (from the challenge query), modify this item to  $(y, c)$  and return  $c$ .
  - Otherwise, choose  $c \xleftarrow{\$} \mathbb{Z}_q$  and update the list  $\mathcal{L}_{H_4} = \mathcal{L}_{H_4} \cup \{(y, c)\}$ , then return  $c$
- (5) The random oracles  $H_1, H_3$  are programmed in the standard way.
- (6) To run the NI-DKG protocol,  $\mathcal{B}$  chooses random polynomials for honest members except the last member:  $f_{m+1}, \dots, f_{n-1}$ . For the  $n$ -th member,  $\mathcal{B}$  chooses randoms  $s_{n,1}, \dots, s_{n,t-1}$  and sets polynomial  $f_n$  to have the values  $f_n(0) = \alpha, f_n(1) = s_{n,1}, \dots, f_n(t-1) = s_{n,t-1}$ . Note that  $\mathcal{B}$  does not know the value of  $\alpha$ , therefore  $\mathcal{B}$  is not able to compute the co-efficients of  $f_n(x)$  or the shares  $s_{n,t} = f_n(t), \dots, s_{n,n} = f_n(n)$ . However,  $\mathcal{B}$  can compute the public parameters  $\text{pp}_n = (g_1^{s_{n,1}}, g_1^{s_{n,2}}, \dots, g_1^{s_{n,t-1}}, g_1^{s_{n,t}}, \dots, g_1^{s_{n,n}}, g_1^\alpha, g_2^\alpha)$ , where  $g_1^{s_{n,j}}$  for  $t \leq j \leq n$  are derived from

$$g_1^{s_{n,j}} = g_1^{\sum_{i \in T} f_n(i) \lambda_{j,i,T}} = g_1^{\alpha \lambda_{j,0,T}} g_1^{\sum_{i \in T \setminus \{0\}} f_n(i) \lambda_{j,i,T}}$$

and  $T = \{0, 1, \dots, t-1\}$ . In the encryption of shares,  $s_{n,t}, \dots, s_{n,n}$  are replaced with randoms.

Let  $\text{QUAL}$  be the non-disqualified nodes at the end of NI-DKG protocol. Since  $\mathcal{B}$  has all the shares generated by the corrupted members,  $\mathcal{B}$  can recover the polynomials  $f_1(x), \dots, f_m(x)$  generated by  $\mathcal{A}$ . The global public key is  $\text{gpk} = g_2^\alpha g_2^{\sum_{i \in \text{QUAL} \setminus \{n\}} f_i(0)}$  where  $\mathcal{B}$  knows the values  $\{f_i(0)\}_{i \in \text{QUAL} \setminus \{n\}}$ .

- The corrupted members that are included in  $\text{QUAL}$  have the shares  $\text{sk}_i = \sum_{j \in \text{QUAL}} f_j(i)$  and  $\text{vk}_i = g_1^{\text{sk}_i}$  for  $i \in \text{QUAL} \cap C$ .
- For honest members  $i \in [m+1, t-1]$ ,  $\text{sk}_i$  and  $\text{vk}_i$  can be computed similarly.
- For honest members  $i \in [t, n]$ ,  $\text{sk}_i$  cannot be computed by  $\mathcal{B}$ , but  $\text{vk}_i$  can be derived as  $\text{vk}_i = \prod_{j \in \text{QUAL}} g_1^{s_{j,i}}$ .
- (7) On an evaluation query  $(\text{Eval}, x, i)$  for an honest  $i$ , invoke random oracle  $H_2$  to get  $H_2(x) = (x, r, h)$ 
  - (a) if  $r \neq \perp$ , return  $\text{vk}_i^r$  with a simulated proof
  - (b) if  $r = \perp$ , return  $\perp$
- (8) On the challenge query  $(\text{Challenge}, x^*, \{(i, z_i^*, \pi_i)\}_{i \in U, V})$ , where  $|V| \geq t$  and  $V \subseteq \text{QUAL}$  and  $U \subseteq V \cap C$  and  $z_i^*$  a set of evaluation shares from the corrupted nodes, is answered as follows:
  - (a) If  $x^*$  was not the  $\eta^*$ -th query to  $H_2$ , then  $\mathcal{B}$  aborts.
  - (b) Otherwise do as follows:
    - (i) Check if  $\pi_i$  is valid for  $i \in U$ . If any check fails, output  $\perp$  and stop.
    - (ii) If there exists  $(y, c)$  in  $H_4$  such that  $e(g_1^\beta, \text{gpk}) = e(y, g_2)$ , then set  $v^* = c$ . Otherwise choose  $v^* \xleftarrow{\$} \mathbb{Z}_q$  and update  $\mathcal{L}_{H_4} = \mathcal{L}_{H_4} \cup \{(\perp, v^*)\}$ . Depending on  $b$  do as follows:
      - (A) If  $b = 0$  then return  $v^*$ ;
      - (B) Else return a uniform random.

- (9) Continue answering evaluation queries as before. If  $\mathcal{A}$  makes queries of the form  $(\text{Eval}, x^*, \cdot)$  then return  $\perp$ .
- (10) Receive a guess  $b'$  from  $\mathcal{A}$ .

The probability that  $\mathcal{B}$  does not abort is  $1/q_{H_2}$  since  $\eta^*$  is uniformly and randomly chosen. Let's consider the case when abort does not happen. In this case, the challenge plaintext  $x^*$  is also the  $\eta^*$ -th distinct query to  $H_2$  and  $H_2(x^*) = g_1^\beta$ . The adversary is not allowed to query  $(\text{Eval}, x^*, \cdot)$  due to the definition of standard pseudorandomness. In other words, the Step 7b will never be executed. In this case  $\mathcal{B}$  simulates the standard pseudorandomness game perfectly from  $\mathcal{A}$ 's point of view.

We define event  $E$  as when the adversary queries  $y^*$  to  $H_4$  such that  $y^* \neq \perp$  and  $e(g_1^\beta, \text{gpk}) = e(y^*, g_2)$ . When  $E$  happens,  $\mathcal{B}$  outputs  $y/(g_1^\beta)^{\sum_{i \in \text{QUAL} \setminus \{n\}} f_i(0)}$  as a solution to the co-CDH problem. From  $\mathcal{A}$ 's point of view, the cases  $b = 0$  and  $b = 1$  are exactly the same unless  $E$  happens. We have that  $\mathcal{A}$ 's advantage  $\text{Adv} = |\Pr[b' = 1 | b = 0, \neg \text{abort}] - \Pr[b' = 1 | b = 1, \neg \text{abort}]|$  of distinguishing  $b = 0$  and  $b = 1$  is not bigger than  $\Pr[E | \neg \text{abort}]$ , i.e.,  $\text{Adv} \leq \Pr[E | \neg \text{abort}]$ . Therefore  $\mathcal{B}$  outputs  $y^*$  as a solution to the co-CDH problem with non-negligible probability  $\Pr[E] \geq \Pr[E | \neg \text{abort}] / q_{H_1}$ .  $\square$

**THEOREM 6.2.**  $\mathcal{V}^{\text{NI-DVRF}}$  achieves strong pseudorandomness under the co-CDH assumption and extended XDH assumption in the random oracle model.

**PROOF.** Similarly to the proof for standard pseudorandomness, we construct hybrids  $\text{Hyb}_{\mathcal{A}}^{\text{snark}}(b)$ ,  $\text{Hyb}_{\mathcal{A}}^{\text{enc}}(b)$  and  $\text{Hyb}_{\mathcal{A}}^{\text{sim}}(b)$ . The arguments for the indistinguishability between these hybrids are similar as before since the partial evaluation queries on the challenge plaintext can be easily answered.

For any adversary  $\mathcal{A}$  that asks  $H_2$  queries on  $q_{H_2}$  distinct  $x$ , we construct hybrid  $\text{Hyb}_{\mathcal{A}}^{\text{fix}}(b)$  in the same way as  $\text{Hyb}_{\mathcal{A}}^{\text{sim}}(b)$  except

- The challenger chooses an index  $\eta \xleftarrow{\$} [q_{H_2}]$  uniformly at random at the beginning of the game.
- On the challenge query  $(\text{Challenge}, x^*, \dots)$ , if  $x^*$  is not the  $\eta$ -th distinct query to  $H_2$ , the challenger aborts.

Since  $\eta$  is uniformly and independently chosen, the probability that abort does not happen is  $1/q_{H_2}$ .  $\text{Hyb}_{\mathcal{A}}^{\text{sim}}(b)$  and  $\text{Hyb}_{\mathcal{A}}^{\text{fix}}(b)$  are exactly the same from the attacker's point of view when abort does not happen. Hence,  $\Pr[\text{Hyb}_{\mathcal{A}}^{\text{sim}}(b) = 1] = q_{H_2} \Pr[\text{Hyb}_{\mathcal{A}}^{\text{fix}}(b) = 1]$ .

Next we construct  $\text{Hyb}_{\mathcal{A}}^{\text{rand}}(b)$  in the exact same way as  $\text{Hyb}_{\mathcal{A}}^{\text{fix}}(b)$  except that

- Let  $C = \{1, \dots, m\}$  and the final polynomial constructed in NI-DKG be  $f$ . If  $m = t-1$ , the rest of the experiment is exactly the same as  $\text{Hyb}_{\mathcal{A}}^{\text{fix}}(b)$ . If  $m < t-1$ , then choose a random polynomial  $f^*$  such that  $f'(0) = f(0), f'(1) = f(1), \dots, f'(m) = f(m)$ .
- On an evaluation query  $(\text{Eval}, x^*, i)$  where  $x^*$  is the  $\eta^*$ -th distinct query to  $H_2$  and  $i \in [m+1, n]$ , compute  $z_i = H_2(x^*)^{f'(i)}$  and a simulated ZK proof  $\pi$ . Return  $(z_i, \pi)$ .

We shall first prove that  $\text{Hyb}_{\mathcal{A}}^{fix}(b)$  and  $\text{Hyb}_{\mathcal{A}}^{rand}(b)$  are indistinguishable under the XDH assumption. Then we will show  $\text{Hyb}_{\mathcal{A}}^{rand}(0)$  and  $\text{Hyb}_{\mathcal{A}}^{rand}(1)$  are indistinguishable under co-CDH assumption.

**$\text{Hyb}_{\mathcal{A}}^{fix}(b)$  and  $\text{Hyb}_{\mathcal{A}}^{rand}(b)$  are indistinguishable.** Suppose there exists an adversary  $\mathcal{A}$  that distinguishes  $\text{Hyb}_{\mathcal{A}}^{rand}(b)$  and  $\text{Hyb}_{\mathcal{A}}^{fix}(b)$ , then we can construct an adversary  $\mathcal{B}$  that breaks the extended XDH assumption using  $\mathcal{A}$  as a subroutine.

Given an extended XDH problem

$$(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2, g_1^{\alpha_1}, \dots, g_1^{\alpha_{t-1}}, g_1^{\beta}, y_1, \dots, y_{t-1})$$

with  $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2, \alpha_1, \dots, \alpha_{t-1}, \beta \xleftarrow{\$} \mathbb{Z}_q, y_i = g_1^{\alpha_i \beta}$  for all  $i$  or randoms,

- (1)  $\mathcal{B}$  chooses an index  $\eta \xleftarrow{\$} [q_{H_2}]$  uniformly at random
- (2) The corruption phase is run as defined. Let  $C = \{1, \dots, m\}$ . If  $m = t - 1$ , the two experiments are indistinguishable because they are defined in the exact same way. In the following discussion, we assume  $m < t - 1$ .
- (3) The NI-DKG phase is run as defined except:  $\mathcal{B}$  chooses  $r_0, r_1, \dots, r_m \xleftarrow{\$} \mathbb{Z}_q$  uniformly at random and creates the last random polynomial  $f_n(x)$  such that  $f_n(0) = r_0, f_n(1) = r_1, \dots, f_n(m) = r_m, f_n(m+1) = \alpha_{m+1}, \dots, f_n(t-1) = \alpha_{t-1}$ . Clearly  $\mathcal{B}$  cannot compute coefficients of  $f_n(x)$  as it doesn't know the values  $\alpha_{m+1}, \dots, \alpha_{t-1}$ . But  $\mathcal{B}$  can compute the public parameters for  $f_n(x)$  as  $\text{pp}_n = (S_1, S_2, \dots, S_n, S_0, g_2^{f_n(0)})$  where

- For  $i \in [0, m], S_i = g_1^{f_n(i)}$
- For  $i \in [m+1, t-1], S_i = g_1^{\alpha_i}$
- For  $i \in [t, n], S_i = \prod_{j \in T} S_j^{\lambda_{i,j,T}}$  for  $T = \{0, 1, \dots, t-1\}$ .

$\mathcal{B}$  can derive the coefficients of  $f_i(x)$  for  $i \in C$  which are created by the adversary. Other polynomials  $f_i(x)$  for  $i \in [m+1, n-1]$  are chosen by  $\mathcal{B}$  randomly. Let QUAL be the final qualified set of members which contains all the honest members and some of the corrupted members. Let  $f = \sum_{i \in \text{QUAL}} f_i$  be the final combined polynomial.

The global public key is  $\text{gpk} = g_1^{\sum_{i \in \text{QUAL}} f_i(0)}$ . The secret keys  $\text{sk}_i$  and verification keys  $\text{vk}_i$  are

- For  $i \in \text{QUAL} \cap C, \text{sk}_i = \sum_{j \in \text{QUAL}} f_j(i), \text{vk}_i = g_1^{\text{sk}_i}$
- For  $i \in [m+1, n], \mathcal{B}$  cannot compute  $\text{sk}_i$  because  $\alpha_i$  is unknown, but can still compute  $\text{vk}_i$  as  $\text{vk}_i = S_i \cdot g_1^{\sum_{j=0}^{n-1} f_j(i)}$ .

- (4) The random oracle  $H_2$  is answered as follows. On query  $x$ ,
  - If there exists a tuple  $(x, r, h) \in \mathcal{L}_{H_2}$ , output  $h$ .
  - Otherwise,
    - if this is the  $\eta$ -th distinct call, set  $r = \perp$  and  $h = g_1^{\beta}$  where  $g_1^{\beta}$  is from the co-CDH problem.
    - else choose a random  $r \xleftarrow{\$} \mathbb{Z}_q$  and set  $h = g_1^r$ .
    - Update  $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup \{(x, r, h)\}$  and output  $h$ .

Give random oracle access to  $\mathcal{A}$ .

- (5) The random oracles  $H_1, H_3, H_4$  are programmed in the standard way.

- (6) On an evaluation query (Eval,  $x_\eta, i$ ) where  $x_\eta$  is the  $\eta$ -th distinct query to  $H_2$  and  $i \in [m+1, n]$ ,
    - If  $i \in [m+1, t-1]$ , compute  $z_i = y_i \cdot g_1^{\beta \sum_{j=0}^{n-1} f_j(i)}$  and a simulated proof  $\pi$ . Return  $(z_i, \pi)$ .
    - If  $i \in [t, n]$ , compute  $z_i = \prod_{j \in T} z_j^{\lambda_{i,j,T}}$  where  $T = \{0, 1, \dots, t-1\}$  and  $z_j = g_1^{\beta f(j)}$  for  $j \in [0, m]$ . Create a simulated proof  $\pi$ . Return  $(z_i, \pi)$ .
- Other evaluation queries (Eval,  $x, i$ ) with  $x \neq x_\eta$  can be easily answered using  $\text{vk}_i^r$  where  $(x, r, h) \in \mathcal{L}_{H_2}$  and simulated proofs.
- (7) On the challenge query (Challenge,  $x^*, \{(i, z_i^*, \pi_i)\}_{i \in U, V}$ ), where  $|V| \geq t$  and  $V \subseteq \text{QUAL}$  and  $U \subseteq V \cap C$  and  $z_i^*$  a set of evaluation shares from the corrupted nodes,
    - (a) If  $x^*$  was not the  $\eta$ -th query to  $H_2$ , then  $\mathcal{B}$  aborts.
    - (b) Otherwise do as follows:
      - (i) Check if  $\pi_i$  is valid for  $i \in U$ . If any check fails, output  $\perp$  and stop.
      - (ii) Since  $\mathcal{B}$  knows the value of  $f(0)$ ,  $\mathcal{B}$  computes  $\pi^* = H_2(x^*)^{f(0)}$  and queries  $H_4$  to obtain  $v^* = H_4(\pi^*)$ . Depending on  $b$  do as follows:
        - (A) If  $b = 0$  then return  $v^*$ ;
        - (B) Else return a uniform random.
  - (8) Continue answering evaluation queries as before. If  $\mathcal{A}$  makes queries of the form (Eval,  $x^*, \cdot$ ) then return  $\perp$ .
  - (9) Receive a guess  $b'$  from  $\mathcal{A}$ .  $\mathcal{B}$  outputs  $b'$ .

Observe, when  $y = g_1^{\alpha \beta}$ ,  $\mathcal{B}$  simulates  $\text{Hyb}_{\mathcal{A}}^{fix}(b)$  perfectly for  $\mathcal{A}$ . When  $y_i$ s are uniformly random, let  $y_i = g_1^{\tau_i}$  for some unknown  $\tau_i$ . The evaluation queries on (Eval,  $x^*, *$ ) are effectively answered with a random polynomial  $f'$  such that  $f'(0) = f(0), f'(1) = f(1), \dots, f'(m) = f(m), f'(m+1) = \frac{\tau_{m+1}}{\beta} + \sum_{j=0}^{m-1} f_j(m+1), f'(t-1) = \frac{\tau_{t-1}}{\beta} + \sum_{j=0}^{n-1} f_j(t-1)$ , which simulates  $\text{Hyb}_{\mathcal{A}}^{rand}(b)$  perfectly for  $\mathcal{A}$ . Therefore  $|\Pr[\text{Hyb}_{\mathcal{A}}^{fix}(b) = 1] - \Pr[\text{Hyb}_{\mathcal{A}}^{rand}(b) = 1]| \leq \text{Adv}_{e-ddh}$ .

Now we are left to show  $\text{Hyb}_{\mathcal{A}}^{rand}(0)$  and  $\text{Hyb}_{\mathcal{A}}^{rand}(1)$  are indistinguishable to conclude the proof of strong pseudorandomness.

**$\text{Hyb}_{\mathcal{A}}^{rand}(0)$  and  $\text{Hyb}_{\mathcal{A}}^{rand}(1)$  are indistinguishable.** Assuming an adversary  $\mathcal{A}$  that distinguishes  $\text{Hyb}_{\mathcal{A}}^{rand}(0)$  and  $\text{Hyb}_{\mathcal{A}}^{rand}(1)$ , we can construct an adversary  $\mathcal{B}$  that breaks the co-CDH assumption using  $\mathcal{A}$  as a subroutine.

Given a co-CDH problem  $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2, g_1^{\alpha}, g_1^{\beta}, g_2^{\alpha})$  with  $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2, \alpha, \beta \xleftarrow{\$} \mathbb{Z}_q$ .  $\mathcal{B}$ 's goal is to output  $g_1^{\alpha \beta}$ .

- (1)  $\mathcal{B}$  chooses an index  $\eta \xleftarrow{\$} [q_{H_2}]$  uniformly at random
- (2) The corruption phase is run as defined. Let  $C = \{1, \dots, m\}$ . If  $m = t - 1$ , the proof is similar to the one for standard pseudorandomness because the adversary has compromised  $t-1$  members and is not allowed to request evaluation query on  $x^*$ . In the following discussion, we assume  $m < t - 1$ .
- (3) The NI-DKG phase is run as defined except:  $\mathcal{B}$  chooses  $r_1, \dots, r_{t-1} \xleftarrow{\$} \mathbb{Z}_q$  uniformly at random and creates the last random polynomial  $f_n(x)$  such that  $f_n(0) = \alpha, f_n(1) = r_1, \dots, f_n(t-1) = r_{t-1}$ . Of course  $\mathcal{B}$  cannot compute

coefficients of  $f_n(x)$  as it doesn't know the value of  $\alpha$ , but  $\mathcal{B}$  can compute the public parameters for  $f_n(x)$  as  $\text{pp}_n = (S_1, S_2, \dots, S_n, S_0, g_2^\alpha)$  where

- For  $i \in [1, t-1]$ ,  $S_i = g_1^{f_n(i)}$
- $S_0 = g_1^\alpha$
- For  $i \in [t, n]$ ,  $S_i = \prod_{j \in T} S_j^{\lambda_{i,j} T}$ , where  $T = \{0, 1, \dots, t-1\}$ .

$\mathcal{B}$  can derive the coefficients of  $f_i(x)$  for  $i \in C$  which are created by the adversary. Other polynomials  $f_i(x)$  for  $i \in [m+1, n-1]$  are chosen by  $\mathcal{B}$  randomly. Let  $\text{QUAL}$  be the final qualified set of members which contains all the honest members and some of the corrupted members. Let  $f = \sum_{i \in \text{QUAL}} f_i$  be the final polynomial.

The global public key is  $\text{gpk} = g_1^\alpha g_1^{\sum_{i \in \text{QUAL}} f_i(0)}$ . The secret keys  $\text{sk}_i$  and verification keys  $\text{vk}_i$  are

- For  $i \in \text{QUAL} \cap C$ ,  $\text{sk}_i = \sum_{j \in \text{QUAL}} f_j(i)$ ,  $\text{vk}_i = g_1^{\text{sk}_i}$
- For  $i \in [m+1, t-1]$ ,  $\text{sk}_i = \sum_{j \in \text{QUAL}} f_j(i)$ ,  $\text{vk}_i = g_1^{\text{sk}_i}$
- For  $i \in [t, n]$ ,  $\mathcal{B}$  cannot compute  $\text{sk}_i$  because  $\alpha$  is unknown, but can still compute  $\text{vk}_i$  as  $\text{vk}_i = S_i \cdot g_1^{\sum_{j=0}^{n-1} f_j(i)}$ .

- (4) The random oracle  $H_2$  is answered as follows. On query  $x$ ,
- If there exists a tuple  $(x, r, h) \in \mathcal{L}_{H_2}$ , output  $h$ .
  - Otherwise,
    - if this is the  $\eta$ -th distinct call, set  $r = \perp$  and  $h = g_1^\beta$  where  $g_1^\beta$  is from the co-CDH problem.
    - else choose a random  $r \xleftarrow{\$} \mathbb{Z}_q$  and set  $h = g_1^r$ .
    - Update  $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup \{(x, r, h)\}$  and output  $h$ .

Give random oracle access to  $\mathcal{A}$ .

- (5) The random oracle  $H_4$  is programmed as follows: Define a list  $\mathcal{L}_{H_4} = \emptyset$ . For a query on  $y$ ,
- Before NI-DKG is completed, the oracle is answered in the standard way,
    - If there exists  $(y, c)$  in  $\mathcal{L}_{H_4}$ , then return  $c$
    - Otherwise, choose a random  $c \xleftarrow{\$} \mathbb{Z}_q$ , update the list  $\mathcal{L}_{H_4} = \mathcal{L}_{H_4} \cup \{(y, c)\}$ , then return  $c$
  - After NI-DKG is completed, the global public key  $\text{gpk}$  has been established,
    - If  $e(g_1^\beta, \text{gpk}) = e(y, g_2)$  and there exists  $(\perp, c) \in \mathcal{L}_{H_4}$  (from the challenge query), modify this item to  $(y, c)$  and return  $c$ .
    - Otherwise, choose  $c \xleftarrow{\$} \mathbb{Z}_q$  and update the list  $\mathcal{L}_{H_4} = \mathcal{L}_{H_4} \cup \{(y, c)\}$ , then return  $c$
- (6) The random oracles  $H_1, H_3$  are programmed in the standard way.
- (7) On an evaluation query  $(\text{Eval}, x_\eta, i)$  where  $x_\eta$  is the  $\eta$ -th distinct query to  $H_2$  and  $i \in [m+1, n]$ , choose  $z_i \xleftarrow{\$} \mathbb{Z}_q$  uniformly at random, and create a simulated proof  $\pi$ . Return  $(z_i, \pi)$ . Note that we choose  $z_i$  randomly here but it won't be a problem as the adversary is only allowed to query  $(\text{Eval}, x_\eta, *)$  for at most  $t-1 - |C|$  times. Other evaluation queries  $(\text{Eval}, x, i)$  with  $x \neq x_\eta$  can be answered using  $\text{vk}_i^r$  where  $(x, r, h) \in \mathcal{L}_{H_2}$  and simulated proofs.

- (8) On the challenge query (Challenge,  $x^*$ ,  $\{(i, z_i^*, \pi_i)\}_{i \in U, V}$ ), where  $|V| \geq t$  and  $V \subseteq \text{QUAL}$  and  $U \subseteq V \cap C$  and  $z_i^*$  a set of evaluation shares from the corrupted nodes,
- (a) If  $x^*$  was not the  $\eta$ -th query to  $H_2$ , then  $\mathcal{B}$  aborts.
  - (b) Otherwise do as follows:
    - (i) Check if  $\pi_i$  is valid for  $i \in U$ . If any check fails, output  $\perp$  and stop.
    - (ii) If there exists  $(y, c)$  in  $H_4$  such that  $e(g_1^\beta, \text{gpk}) = e(y, g_2)$ , then set  $v^* = c$ . Otherwise choose  $v^* \xleftarrow{\$} \mathbb{Z}_q$  and update  $\mathcal{L}_{H_4} = \mathcal{L}_{H_4} \cup \{(\perp, v^*)\}$ . Depending on  $b$  do as follows:
      - (A) If  $b = 0$  then return  $v^*$ ;
      - (B) Else return a uniform random.
- (9) Continue answering evaluation queries as before. If  $\mathcal{A}$  makes queries of the form  $(\text{Eval}, x^*, \cdot)$  then return  $\perp$ .
- (10) Receive a guess  $b'$  from  $\mathcal{A}$ .

Let  $i_1, i_2, \dots, i_{t-m-1}$  be the indices that the adversary requested for evaluation query on  $x^*$  and  $z_{i_1} = g_1^{\tau_{i_1}}, z_{i_2} = g_1^{\tau_{i_2}}, \dots, z_{i_{t-m-1}} = g_1^{\tau_{i_{t-m-1}}}$  with some unknown randoms  $\tau_{i_1}, \dots, \tau_{i_{t-m-1}}$ . It is easy to see that the polynomial used to answer evaluation query on  $x^*$  is effectively a random polynomial  $f'$  such that  $f'(0) = f(0)$ ,  $f'(1) = f(1), \dots, f'(m) = f(m)$ ,  $f'(i_1) = \tau_{i_1}/\beta, \dots, f'(i_{t-m-1}) = \tau_{i_{t-m-1}}/\beta$  (these  $t$  points that uniquely determines  $f'$  with degree  $t-1$ ). Note that, if the adversary does not query at all or query less than  $t-m-1$  times, then we can simply choose randoms to construct  $f'$  implicitly. Hence, the experiment simulates  $\text{Hyb}_{\mathcal{A}}^{\text{rand}}(b)$  perfectly for  $\mathcal{A}$ .

We define event  $E$  as when the adversary queries  $y^*$  to  $H_4$  such that  $y^* \neq \perp$  and  $e(g_1^\beta, \text{gpk}) = e(y^*, g_2)$ . When  $E$  happens,  $\mathcal{B}$  outputs  $y^*/(g_1^\beta)^{\sum_{i \in \text{QUAL} \setminus \{n\}} f_i(0)}$  as a solution to the co-CDH problem. From  $\mathcal{A}$ 's point of view, the cases  $b = 0$  and  $b = 1$  are exactly the same unless  $E$  happens. We have that  $\mathcal{A}$ 's advantage  $\text{Adv} = |\Pr[\text{Hyb}_{\mathcal{A}}^{\text{rand}}(0) = 1] - \Pr[\text{Hyb}_{\mathcal{A}}^{\text{rand}}(1) = 1]| \leq \text{Adv}_{\text{co-CDH}}$ .  $\square$

## 7 CONCLUSION

In this paper we introduced the so far most efficient direct construction and implementation of a non-interactive distributed verifiable random function (NI-DVRF) that is fully compatible with Ethereum. Our NI-DVRF protocol consists of a non-interactive distributed key generation (NI-DKG) and a non-interactive randomness generation. The NI-DKG is constructed using SNARKs and we provide an optimised implementation in Halo2 proof systems. The randomness generation is based on BLS signatures. We formalise and prove the standard and strong pseudorandomness of our NI-DVRF construction in the random oracle model. An Ethereum-compatible implementation of our NI-DVRF protocol on the BN256 curve is currently under investigation for possible deployment on the Boba Network to power the DRB service zkRand. Therefore, our work details several non-trivial optimisations aiming to improve efficiency of the implemented scheme and minimise gas costs for on-chain verification on Ethereum. The experimental evaluations show that our protocol can be easily executed on off-the-self computers. It is practical and cost-effective for real world deployment considering proving time, memory usage and low on-chain verification costs.

## REFERENCES

- [1] Drand. <https://drand.love/about/#about>.
- [2] EIP-1108. <https://eips.ethereum.org/EIPS/eip-1108>.
- [3] Grumpkin curve. <https://hackmd.io/@aztec-network/ByzgNxBfd#2-Grumpkin---A-curve-on-top-of-BN-254-for-SNARK-efficient-group-operations>.
- [4] Halo2. <https://github.com/privacy-scaling-explorations/halo2>.
- [5] Halo2wrong. <https://github.com/privacy-scaling-explorations/halo2wrong>.
- [6] Statement regarding the public report on the analysis of MinRoot, 2023. <https://ethresear.ch/t/statement-regarding-the-public-report-on-the-analysis-of-minroot/16670>.
- [7] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The Oracle Diffie-Hellman Assumptions and an Analysis of DHIES. CT-RSA, page 143–158, 2001.
- [8] Shashank Agrawal, Payman Mohassel, Pratyay Mukherjee, and Peter Rindal. DiSE: Distributed Symmetric-key Encryption. In *CCS 2018*, pages 1993–2010. ACM, 2018.
- [9] L. Baird, S. Garg, A. Jain, P. Mukherjee, R. Sinha, M. Wang, and Y. Zhang. Threshold signatures in the multiverse. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1454–1470, 2023.
- [10] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *SIGACT News*, 15(1):23–27, jan 1983.
- [11] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *ASIACRYPT 2001*, volume 2248, pages 514–532, 2001.
- [12] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In *EUROCRYPT 2005*, pages 302–321, 2005.
- [13] Ignacio Cascudo and Bernardo David. SCRAPE: scalable randomness attested by public entities. In *ACNS*, pages 537–556, 2017.
- [14] David Cash, Eike Kiltz, and Victor Shoup. The Twin Diffie-Hellman Problem and Applications. In *Advances in Cryptology – EUROCRYPT 2008*, pages 127–145, 2008.
- [15] Alisa Cherniaeva, Ilia Shirobokov, and Omer Shlomovits. Homomorphic encryption random beacon. *IACR Cryptol. ePrint Arch.*, 2019:1320, 2019.
- [16] Kevin Choi, Arasu Arun, Nirvan Tyagi, and Joseph Bonneau. Bicorn: An optimistically efficient distributed randomness beacon. In Foteini Baldimtsi and Christian Cachin, editors, *FC 2023*, volume 13950, pages 235–251, 2023.
- [17] Kevin Choi, Aathira Manoj, and Joseph Bonneau. SoK: Distributed Randomness Beacons. In *IEEE S&P 2023*, pages 75–92. IEEE, 2023.
- [18] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain. In *EUROCRYPT 2018*, volume 10821, pages 66–98, 2018.
- [19] Paul Dworkowski. A note on committee random number generation, commit-reveal, and last-revealer attacks. [http://paul.oemm.org/commit\\_reveal\\_subcommittees.pdf](http://paul.oemm.org/commit_reveal_subcommittees.pdf).
- [20] Team Elrond. Elrond: A highly scalable public blockchain via adaptive state sharding and secure proof of stake, 2019. <https://elrond.com/assets/files/elrond-whitepaper.pdf>.
- [21] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, Paper 2019/953, 2019.
- [22] Nicolas Gailly. <https://github.com/nikkolasg/ark-dkg>.
- [23] David Galindo, Jia Liu, Mihai Ordean, and Jin-Mann Wong. Fully distributed verifiable random functions and their application to decentralised random beacons. In *IEEE European Symposium on Security and Privacy, EuroS&P*, pages 88–102. IEEE, 2021.
- [24] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinyu Zhang. hints: Threshold signatures with silent setup. *Cryptology ePrint Archive*, Paper 2023/567, 2023. <https://eprint.iacr.org/2023/567>.
- [25] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. *J. Cryptology*, 20(1):51–83, 2007.
- [26] C. C. F. Pereira Geovandro, Marcos A. Simplicio Jr., Michael Naehrig, and Paulo S. L. M. Barreto. A family of implementation-friendly BN elliptic curves. *Journal of Systems and Software*, 84(8):1319–1326, 2011.
- [27] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. *SOSP '17*, pages 51–68, 2017.
- [28] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT (2)*, pages 305–326. Springer, 2016.
- [29] Jens Groth. Non-interactive distributed key generation and key resharing. *Cryptology ePrint Archive*, Paper 2021/339, 2021. <https://eprint.iacr.org/2021/339>.
- [30] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Aggregatable distributed key generation. page 147–176, 2021.
- [31] Timo Hanke, Mahnush Movahedi, and Dominic Williams. DFINITY technology overview series, consensus system. *CoRR*, abs/1805.04548, 2018.
- [32] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynikov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 357–388, 2017.
- [33] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *FOCS '99*, pages 120–130. IEEE Computer Society, 1999.
- [34] Krzysztof Pietrzak. Simple Verifiable Delay Functions. In *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*, pages 60:1–60:15, 2018.
- [35] Y. Qian. RANDAO: Verifiable Random Number Generation, 2017.
- [36] Adi Shamir. How to share a secret. *Commun. ACM*, 11 1979.
- [37] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *2017 IEEE Symposium on Security and Privacy, SP 2017*, pages 444–460. IEEE Computer Society, 2017.
- [38] Benjamin Wesolowski. Efficient verifiable delay functions. *J. Cryptol.*, 33(4):2113–2147, oct 2020.

## A NON-INTERACTIVE ZERO-KNOWLEDGE ARGUMENTS OF KNOWLEDGE

Given a relation  $R$  with statement and witness  $(x, w)$ , a *non-interactive argument* for  $R$  is a tuple of probabilistic polynomial algorithms (Setup, Prove, Verify) such that:

- $\text{Setup}(R)$ : The setup outputs a proving key  $pk$ , a verification key  $vk$  and a simulation trapdoor  $\tau$ .
- $\text{Prove}(R, pk, x, w)$ : The prover algorithm outputs an argument  $\pi$  for  $(x, w) \in R$ .
- $\text{Verify}(R, vk, x, \pi)$ : The verification algorithm checks the proof  $\pi$  and outputs 1 if valid or 0 otherwise.

We say (Setup, Prove, Verify) is a perfect non-interactive zero knowledge argument of knowledge for  $R$  if it has perfect completeness, perfect zero knowledge and computational knowledge soundness as defined below:

- Perfect completeness: given any true statement, an honest prover should be able to convince an honest verifier, that is, for all  $(x, w) \in R$ ,

$$\Pr[(pk, vk, \tau) \leftarrow \text{Setup}(R); \pi \leftarrow \text{Prove}(R, pk, x, w) : \text{Verify}(R, vk, x, \pi) = 1] = 1$$

- Perfect zero-knowledge: the verifier does not learn any additional information about  $w$  beside the truth of the statement. Formally, there exists a simulator  $\text{Sim}(R, \tau, x)$  such that for all  $(x, w) \in R$  and all adversaries  $\mathcal{A}$ ,

$$\begin{aligned} \Pr[(pk, vk, \tau) \leftarrow \text{Setup}(R); \pi \leftarrow \text{Prove}(R, pk, x, w) : \mathcal{A}(R, pk, vk, \tau, \pi) = 1] \\ = \Pr[(pk, vk, \tau) \leftarrow \text{Setup}(R); \pi \leftarrow \text{Sim}(R, \tau, x) : \mathcal{A}(R, pk, vk, \tau, \pi) = 1] \end{aligned}$$

- Computational knowledge soundness: there is an extractor that can compute a witness whenever the adversary produces a valid argument. For all adversaries  $\mathcal{A}$ , there exists an extractor  $\mathcal{E}_{\mathcal{A}}$  such that,

$$\Pr[(pk, vk, \tau) \leftarrow \text{Setup}(R); ((x, \pi); w) \leftarrow (\mathcal{A} \parallel \mathcal{E}_{\mathcal{A}})(R, pk, vk) : (x, w) \notin R \text{ and } \text{Verify}(R, vk, x, \pi) = 1] \leq \text{negl}$$

## B NI-DVRF DEFINITIONS OF CORRECTNESS

In the following we give formal definitions for robustness and uniqueness.

Robustness ensures the availability of computing the random function value on any plaintext in an adversarial environment. Robustness has been also called *guaranteed output delivery* (G.O.D.) in recent works [13, 32]:

*Definition B.1 (Robustness).* A NI-DVRF protocol  $\mathcal{V} = (\text{KeyGen}, \text{NIDKG}, \text{PartialEval}, \text{PartialVerify}, \text{Combine}, \text{Verify})$  satisfies *robustness* if for all PPT adversaries  $\mathcal{A}$ , the following experiment outputs 1 with negligible probability.

*Corruption*  $\mathcal{A}$  chooses a collection  $C$  of members to corrupt.

Adversary  $\mathcal{A}$  acts on behalf of corrupted nodes, while the challenger acts on behalf of the remaining nodes, which behave honestly (namely they follow the protocol specification). The challenger chooses member public keys for honest members and sends them to the adversary. The adversary chooses member public keys for corrupted members and sends them to the challenger.

*Initialization* Challenger and adversary runs the non-interactive distributed key generation protocol  $\text{NIDKG}(1^\lambda, t, n)$ . After this phase, the protocol establishes a qualified set of members  $\text{QUAL}$ . Every (honest) member  $P_j \in \text{QUAL} \setminus C$  obtains a key pair  $(\text{sk}_j, \text{vk}_j)$ . In contrast, (corrupted) members  $P_j \in C$  end up with key pairs  $(\text{sk}_j, \text{vk}_j)$  in which one of keys may be undefined (i.e. either  $\text{sk}_j = \perp$  or  $\text{vk}_j = \perp$ ). At the end of this phase, the global public key  $\text{gpk}$  and the verification keys  $\{\text{vk}_i\}_{i \in \text{QUAL}}$  are known by both the challenger and the adversary.

*Query* In response to  $\mathcal{A}$ 's evaluation query  $(\text{Eval}, x, i)$  for some honest member  $P_i \in \text{QUAL} \setminus C$  and plaintext  $x$ , the challenger returns  $\sigma_x^i \leftarrow \text{PartialEval}(x, \text{sk}_i, \text{vk}_i)$ . In any other case, the challenger returns  $\perp$ .

*Challenge* The challenger receives from  $\mathcal{A}$  a set  $U \subseteq \text{QUAL}$ , of size at least  $t$ , a plaintext  $x^*$  and a set of evaluation shares

$\{(i, v_i, \pi_i)\}_{i \in U \cap C}$  corresponding to members under adversarial control. Challenger proceeds to compute the partial evaluations corresponding to honest nodes as  $(i, v_i, \pi_i) \leftarrow \text{PartialEval}(x^*, \text{sk}_i, \text{vk}_i)$  for  $i \in U \setminus C$ . Let  $(v^*, \pi^*) \leftarrow \text{Combine}(\mathcal{VK}, x^*, \{(i, v_i, \pi_i)\}_{i \in U})$ . Output 1 if  $v^* \neq \perp$  and  $\text{Verify}(\text{gpk}, x^*, v^*, \pi^*) = 0$ ; else, output 0.

Uniqueness guarantees that it is infeasible for any adversary to compute two different output values  $v, v'$  and a plaintext  $x$  such that both values pass the verification test wrt  $x$ , even when the secret keys of the honest nodes are leaked.

*Definition B.2 (Uniqueness).* A NI-DVRF protocol  $\mathcal{V} = (\text{KeyGen}, \text{NIDKG}, \text{PartialEval}, \text{PartialVerify}, \text{Combine}, \text{Verify})$  satisfies *uniqueness* if for all PPT adversaries  $\mathcal{A}$ , the following experiment outputs 1 with negligible probability.

*Corruption and Initialization* these two phases are defined exactly as in Definition B.1 (Robustness).

*Query* The adversary  $\mathcal{A}$  can issue evaluation query and key revealing query.

- In response to  $\mathcal{A}$ 's evaluation query  $(\text{Eval}, x, i)$  for some honest member  $P_i \in \text{QUAL} \setminus C$  and plaintext  $x$ , the challenger returns  $\sigma_x^i \leftarrow \text{PartialEval}(x, \text{sk}_i, \text{vk}_i)$ . In any other case, the challenger returns  $\perp$ .
- In response to  $\mathcal{A}$ 's key revealing query  $(\text{KeyRev}, j)$  for some honest member  $P_j \in \text{QUAL} \setminus C$ , the challenger returns  $\text{sk}_j$ .

*Challenge* The challenger receives from the adversary  $\mathcal{A}$ , a plaintext  $x^*$ , two values  $v, v'$  and two proofs  $\pi, \pi'$ . Output 1 if  $v \neq v'$  and  $\text{Verify}(\text{pk}, x, v, \pi) = \text{Verify}(\text{pk}, x, v', \pi') = 1$ ; else, output 0.