Fully Distributed Verifiable Random Functions and their Application to Decentralised Random Beacons

David Galindo[1,2], Jia Liu[1], Mihai Ordean[2] and Jin-Mann Wong[1]

[1] Fetch.AI, Cambridge, UK
[2] University of Birmingham, UK

Abstract. We provide the first systematic analysis of (Non-Interactive) Fully Distributed Verifiable Random Functions (DVRFs), including their syntax, definition of integrity and privacy properties, and describe and analyse three concrete constructions, two of which are original. Building on recent work (Agrawal, Mohassel, Mukherjee, Rindal: CCS 2018), we strengthen the standard pseudorandomness property by allowing an adversary to make partial queries on the challenge value, and call the resulting property *strong pseudorandomness*. We show how a prominent DVRF construction in the blockchain space meets standard pseudorandomness, and provide two other instantiations that meet strong pseudorandomness, under widely accepted cryptographic assumptions. We review how to generically build a Decentralized Random Beacon (DRB) from any DVRF instance. DRBs have recently gained a lot traction as a key component for leader(s) election in decentralized ledger technologies. We provide implementations and experimental evaluations of three concrete DVRFs, using different cryptographic libraries. Our two new DRB instantiations are strongly pseudorandom and strongly unbiasable, while exhibiting high performance and linear communication complexity (as they are in essence non-interactive). We provide a C++ reference implementation that is available in open source form.

Keywords Cryptography, Blockchain, Random Beacon, Distributed Computation, Leader Election

1 Introduction

In recent years there has been prolific development in blockchain technologies [Nak08,Woo19], and a plethora of platforms relying on blockchains have seen the light of day. The various platforms often differ in their design choices, and thus on the consensus protocol they rely on. Many consensus protocols involve allocating the creation of blocks with a block producer, whose selection procedure most often than not[GHM$^+$17,BKM18,DGKR18,HMW18,KJG$^+$18] requires a method for collective randomness sampling. In order to avoid reliance on a trusted party, a common approach is to use a mechanism that allows the distributed computation of an unpredictable and unbiased source of randomness, verifiably.

Verifiable Random Function (VRF) This primitive was introduced by Micali, Rabin and Vadhan [MRV99] and can be seen as the public-key version of a keyed cryptographic hash $F_{\mathsf{sk}}(\cdot)$, where a trusted party evaluates $F_{\mathsf{sk}}(x)$ on inputs x in such a way that the output can publicly be verified for correctness via an auxiliary proof π_x. The secret key sk allows i) evaluation of $F_{\mathsf{sk}}(\cdot)$ on any input x and ii) to compute and provide proof π_x of the correct evaluation $F_{\mathsf{sk}}(x)$. The proof correctness can be verified by means of an algorithm Verify that takes as inputs the public key pk corresponding to sk, the values x, $F_{\mathsf{sk}}(x)$, and proof π_x, and outputs accept or reject.

Unique Signatures There are similarities between a VRF and a digital signature scheme with unique signatures. On inputs a string x and a secret signing key SK_S, a signing algorithm outputs a signature σ_x. If the signature scheme is unforgeable, the latter value σ_x is unpredictable given the public key PK_S. Despite its unpredictability, the signature σ_x on string x is *publicly verifiable*. There are two issues that may prevent signature schemes with unique signatures from being used straightforwardly as a VRF: i) signatures σ_x may not be unique given x and PK_S; and ii) σ_x is unpredictable but *not* pseudorandom (e.g. signatures could contain some bias and be distinguishable from a random distribution). Needless to say, most secure signature schemes (e.g. ECDSA, Schnorr, PSS-RSA, qTESLA) are either probabilistic or history dependent, i.e. stateful (e.g. eXtended Merkle Signature Scheme). Both these properties are in conflict with the uniqueness requirement. It is hence necessary that the verification procedure accepts a unique signature for any given message, for a fixed public key. A signature scheme living up to those criteria is said to have *unique signatures* [GO93,FZ12]. VRFs derived from unique signatures present strong unbiasibility properties due to the uniqueness, even in the presence of active adversaries, of the corresponding pseudorandom value.

Our contributions. We provide the first systematic analysis of (Non-Interactive) Fully Distributed Verifiable Random Functions (DVRFs), including their syntax and the definition of their integrity and privacy properties. We extend [AMMR18] and define *standard and strong pseudorandomness* for DVRFs, where the latter strengthens the standard pseudorandomness property by allowing an adversary to make partial queries on the target pseudorandom value (Section 3). We validate the security of Dfinity-DVRF, a prominent DVRF construction in the blockchain space [HMW18], by showing that it meets standard pseudorandomness (Section 5) and discuss how standard proof techniques [Bol02,GJKR07] fail to extend to prove strong pseudorandomness.

We review how to to generically build a Decentralized Random Beacon (DRB) from any secure DVRF (Section 6). Additionally, we provide *two new instantiations* that meet strong pseudorandomness, under widely accepted cryptographic assumptions. One of these constructions is called DDH-DVRF and uses standard elliptic curve cryptography (Section 4), whereas the other construction is named GLOW-DVRF and uses cryptographic pairings (Section 5). By applying the generic DVRF-to-DRB transformation, we obtain two new DRBs designs that enjoy strong pseudorandomness and strong unbiasability. We note that the proof of correctness π_x in DDH-DVRF is non-compact, i.e., its size is linear in the threshold t, while Dfinity-DVRF and GLOW-DVRF are able to provide compact proof, i.e., constant-size.

We provide an experimental evaluation of GLOW-DVRF, DDH-DVRF and Dfinity-DVRF, using the cryptographic libraries MCL [Mis19], RELIC [AG14] and Libsodium [BD19b]. Our experiments show that both GLOW-DVRF and DDH-DVRF outperform Dfinity-DVRF in running time, approximately by x3 and x5 respectively at the 128-bit security level. We provide a reference implementation in C++ and make it widely available as open source code [Fet20] (Apache 2.0 license).

1.1 Related Work

To our knowledge, the first distributed VRF construction was proposed by Dodis [Dod03] and required the existence of a trusted dealer. The constructions described in this work dispose of this trusted dealer by using a Distributed Key Generation (DKG) sub-protocol: Dfinity-DVRF and DDH-DVRF use a protocol by Gennaro, Jarecki, Krawczyk and Rabin [GJKR07], whereas GLOW-DVRF required several modifications to the latter. Manulis and Kuchta [KM13] proposed a generic construction for interactive distributed VRFs based on unique aggregate signatures in the shared random string model. Compared to our DVRF syntax and new designs, the concrete constructions obtained from [KM13] are at least two orders of magnitude less efficient than ours,

both in running time (as they use pairings and an inefficient generic transformation of pseudorandom functions from unpredictable functions [MRV99]) and in latency (as they involve sequential interaction between a number of peers).

Recently [AMMR18] introduced strong notions of correctness and pseudorandomness for Distributed Pseudorandom Functions in the presence of active adversaries. Our work generalises these definitions to the Verifiable Random Functions scenario. [HMW18] informally introduced the influential Dfinity-DVRF (used e.g. in [Cor19,Kee19,Clo19,DAO19,SJSW19]) with no formal security model nor analysis. We formalize Dfinity-DVRF and prove its security wrt to standard pseudorandomness under the co-CDH assumption in pairing groups in the random oracle model. We discuss how the techniques [Bol02,GJKR07] that have been used to prove strong unforgeability of threshold BLS signatures [BLS01] do not trivially allow to prove strong pseudorandomness. In contrast, our new GLOW-DVRF also uses pairing groups, but achieves strong pseudorandomess in the random oracle model under the co-CDH and eXternal DDH assumptions. Perhaps surprsingly, GLOW-DVRF shows not only stronger security than Dfinity-DVRF but also possesses better running times.

Our construction DDH-DVRF also achieves strong pseudorandomness but under the standard DDH assumption. The main drawback raised in [SJSW20] against DRBs built from DVRFs is the fact that they rely on computational assumptions on pairing groups, which are known to be easier hard problems to cryptoanalyse than those obtained from ordinary elliptic curves [BD19a]. Our proposed DDH-DVRF resolves this issue as it builds on ordinary elliptic curves.

2 Building Blocks

We recall next the Chaum-Pedersen proof system for equality of discrete logarithms, the syntax of VRFs and the computational assumptions needed to build our concrecte DVRFs constructions.

2.1 Equality of Discrete Logarithms NIZK

We need NIZK proof systems as an ingredient to our construction, namely the *Equality of Discrete Logarithms* proof system. Formally, given a cyclic group $\mathbb{G}$ of order q and $g, h \in \mathbb{G}$, the NIZK proof $(\mathsf{PrEq}_H, \mathsf{VerifyEq}_H)$ to show $k = \log_g x = \log_h y$ for $x, y \in \mathbb{G}$, $k \in \mathbb{Z}_q$ is described as below [CP93]:

- $\mathsf{PrEq}_H(g, h, x, y, k)$ chooses $r \xleftarrow{R} \mathbb{Z}_q$, computes $com_1 = g^r, com_2 = h^r$ and sets $ch \leftarrow H(g, h, x, y, com_1, com_2)$. Output is $(ch, res = r + k \cdot ch)$.
- $\mathsf{VerifyEq}_H(g, h, x, y, ch, res)$ computes $com_1 \leftarrow g^{res}/x^{ch}$ and $com_2 \leftarrow h^{res}/y^{ch}$ and outputs $ch \stackrel{?}{=} H(g, h, x, y, com_1, com_2)$.

2.2 Verifiable Random Function (VRF)

Formally, a Verifiable Random Function (VRF) $\mathcal{V} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Verify})$ consists of the following algorithms [MRV99]:

$\mathsf{KeyGen}(1^\lambda)$ is a *key generation* algorithm that takes as input a security parameter 1^λ; it outputs a public key pk and a secret key sk.

$\mathsf{Eval}(\mathsf{sk}, \mathsf{vk}, x)$ is an *evaluation algorithm* that takes as input secret key sk and verification key vk, a message x, and outputs a triple $(x, F_{\mathsf{sk}}(x), \pi)$, where $F_{\mathsf{sk}}(x)$ is the function's evaluation on input x and π is a non-interactive proof of correctness.

$\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, x, v, \pi)$ is a *verification algorithm* that takes as input the public key pk, a verification key vk, a message x, a value v and a proof π and outputs `accept` or `reject`.

2.3 Assumptions

In the following we recall the definitions of two variants of the Decisional Diffie-Hellman assumption, the Discrete Logarithm assumption and Lagrange coefficients for polynomial interpolation.

Definition 1 (Diffie-Hellman Groups). *Let $\mathbb{G} = \langle g \rangle$ be a (cyclic) group of order q prime. We assume there exists an efficient group instance generator algorithm $\mathcal{I}$ that on input a security parameter 1^λ outputs the description of $\langle \mathbb{G}, q, g \rangle$, with q a λ-bit length prime.*

Definition 2 (DDH assumption). *Let $X \leftarrow \left(\mathbb{G}, q, g, g^a, g^b\right)$ where $\langle \mathbb{G}, q, g \rangle \leftarrow \mathcal{I}(1^\lambda)$ and $a, b \xleftarrow{R} \mathbb{Z}_q^*$. We say that $\mathcal{I}$ satisfies the* Decisional Diffie-Hellman assumption *if for $r \xleftarrow{R} \mathbb{Z}_q^*$ the value*

$$\mathbf{Adv}^{\mathrm{DDH}}_{\mathcal{I},\mathcal{A}}(\lambda) := \left| \Pr\left[\mathcal{A}(X, g^{ab}) = 1\right] - \Pr\left[\mathcal{A}(X, g^r) = 1\right] \right|$$

is negligible in λ. The probabilities are computed over the internal random coins of $\mathcal{A}$, $\mathcal{I}$ and the random coins of the inputs.

Definition 3 (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 $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ to a group $\mathbb{G}_T$ is called a* bilinear *map, if it satisfies the following three properties:*

- *Bilinearity: $\mathbf{e}(g_1^a, g_2^b) = \mathbf{e}(g_1, g_2)^{ab}$ for all $a, b \in \mathbb{Z}_p$.*
- *Non-Degenerate: $\mathbf{e}(g_1, g_2) \neq 1$.*
- *Computable: $\mathbf{e}(g_1, g_2)$ can be efficiently computed.*

We assume there exists an efficient bilinear pairing instance generator algorithm $\mathcal{IG}$ that on input a security parameter 1^λ outputs the description of $\langle \mathbf{e}(\cdot,\cdot), \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q \rangle$.

Asymmetric pairing groups can be efficiently generated [GJNB11]. Pairing group exponentiations and pairing operations can also be efficiently computed [DSD07].

Definition 4 (Computational co-CDH assumption [BDN18]). *Let*

$$X \leftarrow \left(\mathbf{e}(\cdot,\cdot), \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2, g_1^a, g_1^b, g_2^a\right)$$

where $\langle \mathbf{e}(\cdot,\cdot), \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q \rangle \leftarrow \mathcal{IG}(1^\lambda)$ and $a, b \xleftarrow{R} \mathbb{Z}_q^*$ and $g_1 \xleftarrow{R} \mathbb{G}_1, g_2 \xleftarrow{R} \mathbb{G}_2$. We say that $\mathcal{IG}$ satisfies the Computational co-CDH assumption in $\mathbb{G}_1$ *if*

$$\mathbf{Adv}^{\text{co-CDH}}_{\mathcal{IG},\mathcal{A}}(\lambda) := \Pr\left[\mathcal{A}(X) = g_1^{ab}\right]$$

is negligible in λ.

Definition 5 (XDH assumption [CHL05]). *Let $\langle \mathbf{e}(\cdot,\cdot), \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q \rangle \leftarrow \mathcal{IG}(1^\lambda)$ be a a bilinear mapping. The XDH assumption states that DDH is hard in $\mathbb{G}_1$.*

Definition 6 (Lagrange coefficients). *For a key reconstruction set Δ, 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 $\sum_{i \in \Delta} f(i)\lambda_{0,i,\Delta} = f(0)$.*

3 Distributed Verifiable Random Functions: Formal Definitions

Let $a, b : \mathbb{N} \to \mathbb{N}$ be polynomial time functions, and where $a(\lambda), b(\lambda)$ both are bounded by a polynomial in λ. Let F be a function with domain $\mathcal{D}$ and range $\mathcal{R}$. Let $\mathcal{D}$ and $\mathcal{R}$ be sets of size $2^{a(\lambda)}$ and $2^{b(\lambda)}$ respectively. The goal of a DVRF is to initialize a Pseudo-Random function and compute $F_{\mathsf{sk}}(x)$ for inputs x by a set of servers $S_1, \ldots, S_\ell$ with no central party.

3.1 Syntax and Basic Properties

In the Setup phase ℓ servers $S_1, \ldots, S_\ell$ communicate via pairwise private and authenticated channels. They have access to an append-only public board where every server can post messages, and these posts cannot by repudiated by their senders. A setup interaction is then run between the ℓ servers to build a global public key pk, individual servers' secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell$, and individual servers' public verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell$. The servers' secret and verification keys $(\mathsf{sk}_i, \mathsf{vk}_i)$ for $i = 1, \ldots, \ell$ will later enable any subset of $t+1$ servers to non-interactively compute the verifiable random value $F_{\mathsf{sk}}(x)$ on a plaintext $x \in \mathcal{D}$. On the contrary, any set of at most t servers can not learn any information on $F_{\mathsf{sk}}(x)$ for any x not previously computed.

Definition 7 (DVRF). *A t-out-of-ℓ (Non-Interactive) Fully Distributed Verifiable Random Function (DVRF) $\mathcal{V} = (\mathsf{DistKG}, \mathsf{PartialEval}, \mathsf{Combine}, \mathsf{Verify})$ consists of the following algorithms:*

- $\mathsf{DistKG}(1^\lambda, t, \ell)$ *is a fully distributed* key generation *algorithm that takes as input a security parameter 1^λ, the number of participating servers ℓ, and the threshold parameter t; it outputs a set of qualified servers* QUAL*, a public key pk, a list $\{\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell\}$ of servers' secret keys, a list $\mathsf{vk} = \{\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell\}$ of servers' verification keys.*
- $\mathsf{PartialEval}(\mathsf{sk}_i, \mathsf{vk}_i, x)$ *is a* partial evaluation algorithm *that takes as input server $S_i \in$ QUAL, secret key sk_i and verification key vk_i, a plaintext x, and outputs either a triple $s_x^i = (i, v_i, \pi_i)$, where v_i is the i-th evaluation share and π_i is a non-interactive proof of correct partial evaluation.*
- $\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, x, \mathcal{E})$ *is a* combination *algorithm that takes as input the global public key pk, the verification keys vk, a plaintext x, and a set $\mathcal{E} = \{s_x^{i_1}, \ldots, s_x^{i_{|\mathcal{E}|}}\}$ of partial function evaluations originating from $|\mathcal{E}| \geq t+1$ different servers, and outputs either a pair (v, π) of pseudo-random function value v and correctness proof π, or $\perp$.*
- $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, x, v, \pi)$ *is a* verification algorithm *that takes as input the public key pk, a set of verification keys vk, a plaintext x, a value v and a proof π and outputs* `accept` *or* `reject`*.*

Any DVRF must satisfy four basic properties: *consistency*, meaning that no matter which collection of correctly formed shares is used to compute the function on a plaintext x the same random value $v = F_{\mathsf{sk}}(x)$ is obtained; *domain-range correctness*, meaning that every computed value v belongs to the range domain $\mathcal{R}$; *provability*, meaning that the uniquely recovered value $v = F_{\mathsf{sk}}(x)$ passes the verification test; and *uniqueness*, meaning that for every plaintext x a unique value $v = F_{\mathsf{sk}}(x)$ passes the verification test. Formally,

Definition 8 (Admissibility). *A (Non-Interactive) Fully Distributed Verifiable Random Function $\mathcal{V} = (\mathsf{DistKG}, \mathsf{PartialEval}, \mathsf{Combine}, \mathsf{Verify})$ is said to be* admissible *if it satisfies the following properties of* consistency, domain range correctness, provability and uniqueness *with overwhelming probability, namely with probability at least $1 - \mathsf{negl}(\lambda)$ for a negligible function $\mathsf{negl}(\lambda)$:*

- Consistency: *for any integers $0 \leq t < \ell$, for every plaintext $x \in \mathcal{D}$, and any two $(t+1)$-subsets of partial evaluation shares $\mathcal{E} \neq \mathcal{E}'$ obtained by running $\mathsf{PartialEval}(\cdot, \cdot, x)$, it holds that $v = v' (\neq \perp)$, where $(v, \pi) \leftarrow \mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, x, \mathcal{E})$ and $(v', \pi') \leftarrow \mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, x, \mathcal{E}')$.*
- Domain Range Correctness: *for any integers $0 \leq t < \ell$, for every plaintext $x \in \mathcal{D}$, and any $(t+1)$-subset $\mathcal{E}$ of partial evaluation shares such that $\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, x, \mathcal{E}) = (v, \pi) \neq \perp$ it holds that $v \in \mathcal{R}$.*
- Provability: *for any integers $0 \leq t < \ell$, every plaintext $x \in \mathcal{D}$, and any $t+1$-subset $\mathcal{E}$ of partial evaluation shares such that $\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, x, \mathcal{E}) = (v, \pi) \neq \perp$ it holds that $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, x, v, \pi) =$ `accept`.*
- Uniqueness: *no integers t, ℓ with $0 \leq t < \ell$, no plaintext $x \in \mathcal{D}$, no proofs π, π' and no values $v, v' \in \mathcal{R}$ can be found such that $v \neq v'$ and $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, x, v, \pi) = \mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, x, v', \pi') =$ `accept`.*

3.2 Security Properties: Correctness and Pseudorandomness

We give rigorous definitions of *correctness* and *pseudo-randomness* properties against active adversaries, building on [AMMR18]. Roughly speaking, correctness ensures the availability of computing the random function value on any plaintext in an adversarial environment; pseudo-randomness ensures that no adversary controlling at most t servers is able to distinguish the outputs of the function from random.

Definition 9 (Correctness). *A* DVRF *protocol* $\mathcal{V} = (\mathsf{DistKG}, \mathsf{PartialEval}, \mathsf{Combine}, \mathsf{Verify})$ *satisfies* correctness *if for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that the following experiment outputs 1 with probability at most* $\mathsf{negl}(\lambda)$*.*

[Corruption] *On input the Servers list* $S = \{S_1, \ldots, S_\ell\}$ *and threshold* $0 \leq t < \ell$*, a correctness adversary* $\mathcal{A}$ *chooses a collection* C *of servers to be corrupted with* $|C| \leq t$*. Adversary* $\mathcal{A}$ *acts on behalf of corrupted servers, while the challenger acts on behalf of the remaining servers, which behave honestly (namely they follow the protocol specification).*

[Initialization] *Challenger and adversary engage in running the distributed key generation protocol* $\mathsf{DistKG}(1^\lambda, t, \ell)$*. After this phase, the protocol establishes a qualified set of servers* QUAL*. Every (honest) server* $S_j \in \text{QUAL} \setminus C$ *obtains a key pair* $(\mathsf{sk}_j, \mathsf{vk}_j)$*. In contrast, (corrupted) servers* $S_j \in C$ *end up with key pairs* $(\mathsf{sk}_j, \mathsf{vk}_j)$ *in which one of keys may be undefined (i.e. either* $\mathsf{sk}_j = \perp$ *or* $\mathsf{vk}_j = \perp$*). At the end of this phase, the global public key* pk *and the verification keys vector* vk *is known by both the challenger and the attacker.*

[Evaluation] *In response to* $\mathcal{A}$*'s evaluation query* (Eval, x, i) *for some honest server* $S_i \in \text{QUAL} \setminus C$ *and plaintext* $x \in \mathcal{D}$*, the challenger returns* $s_x^i \leftarrow \mathsf{PartialEval}(\mathsf{sk}_i, \mathsf{vk}_i, x)$*. In any other case, the challenger returns* $\perp$*.*

[Computation] *The challenger receives from the adversary* $\mathcal{A}$ *a set* U *of size at least* $t + 1$ *and* $U \subseteq \text{QUAL}$*, a plaintext* $x^\star \in \mathcal{D}$ *and a set of evaluation shares* $\{(i, v_i^\star, \pi_i^\star)\}_{i \in U \cap C}$*. Compute* $(j, v_j, \pi_j) \leftarrow \mathsf{PartialEval}(\mathsf{sk}_j, \mathsf{vk}_j, x^\star)$ *for* $j \in U$ *and* $(j, v_j^\star, \pi_j^\star) \leftarrow \mathsf{PartialEval}(\mathsf{sk}_j, \mathsf{vk}_j, x^\star)$ *for* $j \in U \setminus C$*. Let* $v \leftarrow \mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, x^\star, \{(j, z_j, \pi_j)\}_{j \in U})$ *and* $v^\star \leftarrow \mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, x^\star, \{(i, v_i^\star, \pi_i^\star)\}_{i \in U})$*. Output 0 if* $v^\star \in \{v, \perp\}$*; else, output 1.*

In the above definition the value v is obtained by running the combine function on inputs that were output by calling the partial evaluation function, whereas $v^\star$ is obtained by running the combine function on a set that contains adversarial inputs. *Correctness* demands that an adversary cannot alter the output of $\mathsf{Combine}(\cdot)$ (i.e. making $v^\star \neq v$) other than getting adversarial input rejected.

For strong pseudorandomnes the definition captures the intuition of the real world security explicitly and accurately by allowing an adversary to choose the set of parties to corrupt, obtain partial evaluations from the honest parties on the challenge plaintext (up to the threshold), and participate in computing the pseudorandom function on the challenge plaintext.

Definition 10 (Strong Pseudorandomness [AMMR18]). *A* DVRF $\mathcal{V} = (\mathsf{DistKG}, \mathsf{PartialEval}, \mathsf{Combine}, \mathsf{Verify})$ *is* strongly pseudorandom *if for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that* $\left|\Pr[\mathsf{PseudoRand}_{\mathcal{V},\mathcal{A}}(1^\lambda, 0) = 1] - \Pr[\mathsf{PseudoRand}_{\mathcal{V},\mathcal{A}}(1^\lambda, 1) = 1]\right| = \mathsf{negl}(\lambda)$ *where the experiment* $\mathsf{PseudoRand}_{\mathcal{V},\mathcal{A}}(1^\lambda, b)$ *is defined below:*

[Corruption] *On input the Servers list* $S = \{S_1, \ldots, S_\ell\}$ *and threshold* $0 \leq t < \ell$*, a correctness adversary* $\mathcal{A}$ *chooses a collection* C *of servers to be corrupted with* $|C| \leq t$*. Adversary* $\mathcal{A}$ *acts on behalf of corrupted servers, while the challenger acts on behalf of the remaining servers, which behave honestly (namely they follow the protocol specification).*

[Initialization] *Challenger and adversary engage in running the distributed key generation protocol* $\mathsf{DistKG}(1^\lambda, t, \ell)$. *After this phase, the protocol establishes a qualified set of servers* QUAL. *Every (honest) server* $S_j \in \mathrm{QUAL} \setminus C$ *obtains a key pair* $(\mathsf{sk}_j, \mathsf{vk}_j)$. *In contrast, (corrupted) servers* $S_j \in C$ *end up with key pairs* $(\mathsf{sk}_j, \mathsf{vk}_j)$ *in which one of keys may be undefined (i.e. either* $\mathsf{sk}_j = \perp$ *or* $\mathsf{vk}_j = \perp$*). At the end of this phase, the global public key* pk *and the verification keys vector* vk *is known by both the challenger and the attacker.*

[Pre-Challenge Evaluation] *In response to* $\mathcal{A}$*'s evaluation query* (Eval, x, i) *for some honest server* $S_i \in \mathrm{QUAL} \setminus C$ *and plaintext* $x \in \mathcal{D}$, *the challenger returns* $s_x^i \leftarrow \mathsf{PartialEval}(\mathsf{sk}_i, \mathsf{vk}_i, x)$. *In any other case, the challenger returns* $\perp$.

[Challenge] *The challenger receives from the adversary* $\mathcal{A}$ *a set of evaluation shares* $\{s_{x^\star}^i\}_{S_i \in U \cap C}$ *with* $U \subseteq \mathrm{QUAL}$ *and* $|U| \geq t+1$, *and a plaintext* $x^\star \in \mathcal{D}$, *such that* $(\mathsf{Eval}, x^\star, i)$ *has been queried at most* $t - |C|$ *times for different* $S_i \in \mathrm{QUAL} \setminus C$. *Let* $s_{x^\star}^j \leftarrow \mathsf{PartialEval}(\mathsf{sk}_j, \mathsf{vk}_j, x^\star)$ *for* $S_j \in U \setminus C$ *and* $(v^\star, \pi^\star) \leftarrow \mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, x^\star, \{s_{x^\star}^j\}_{S_j \in U})$. *If* $v^\star = \perp$ *the experiment output* $\perp$. *Otherwise, if* $b = 0$ *the adversary receives* $v^\star$*; if* $b = 1$ *the adversary receives a uniform random value in* $\mathcal{R}$.

[Post-Challenge Evaluation] *In response to* $\mathcal{A}$*'s evaluation query* (Eval, x, i) *with* $x \neq x^\star$ *for some server* $S_i \in \mathrm{QUAL} \setminus C$ *and plaintext* $x \in \mathcal{D}$, *the challenger returns* $s_x^i \leftarrow \mathsf{PartialEval}(\mathsf{sk}_i, \mathsf{vk}_i, x)$. *In any other case, the challenger returns* $\perp$.

When the adversary is not allowed to obtain any partial evaluation on the challenge plaintext in the above experiment, we refer to it as *standard pseudorandomness*. This weaker notion of pseudorandomness has been the standard up to now in the related literature on Distributed Random Verifiable Functions [DY05,BLMR13,KM13]. The usage of this weaker definition of pseudorandomness can also be found in the DRB literature e.g. [CD17].

4 DDH-based DVRF with non-compact proofs

In this section we give full descriptions and proofs of a fully distributed non-interactive verifiable random function based on the Decisional Diffie-Hellman problem. The construction is obtained by using the distributed key generation protocol by Gennaro, Jarecki, Krawczyk and Rabin [GJKR07] in the setup phase and the DDH-based VRF presented in [MBB+15,GRPV18].

Let $(\mathbb{G}, g, q)$ be a mutiplicative group where the DDH assumption holds. Let $H_1 : \{0,1\}^* \to \mathbb{G}$ and $H_2 : \{0,1\}^* \to \mathbb{Z}_q$ be hash functions. Let $\mathcal{V}^{\mathsf{DDH\text{-}DVRF}} = (\mathsf{DistKG}, \mathsf{PartialEval}, \mathsf{Combine}, \mathsf{Verify})$ be the DVRF defined as follows:

$\mathsf{DistKG}(1^\lambda, t, \ell)$ is run by ℓ participating severs $S = \{S_1, \ldots, S_\ell\}$. Each server S_i chooses a random polynomial $f_i(z) = a_{i,0} + a_{i,1}z + \cdots + a_{i,t}z^t$. The protocol outputs a set of qualified servers $\mathrm{QUAL} \subseteq S$, a secret key $\mathsf{sk}_i = \sum_{j \in \mathrm{QUAL}} f_j(i) \in \mathbb{Z}_q$ and a verification key $\mathsf{vk}_i = g^{\mathsf{sk}_i} \in \mathbb{G}$ for each $i \in \mathrm{QUAL}$, an implicit distributed secret value $\mathsf{sk} = \sum_{i \in \mathrm{QUAL}} a_{i,0} \in \mathbb{Z}_q$, and a global public key $\mathsf{pk} = \prod_{i \in \mathrm{QUAL}} g^{a_{i,0}}$. The full description of the protocol can be found in Figure 1 in Appendix A.

$\mathsf{PartialEval}(\mathsf{sk}_i, \mathsf{vk}_i, x)$ outputs $s_x^i = (i, v_i, \pi_i)$ for a plaintext x, where $v_i \leftarrow H_1(x)^{\mathsf{sk}_i}$ and $\pi_i \leftarrow \mathsf{PrEq}_{H_2}(g, \mathsf{vk}_i, H_1(x), v_i; r)$ for randomness $r \xleftarrow{R} \mathbb{Z}_q$.

$\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, x, \mathcal{E})$ parses list $\mathcal{E} = \{s_x^{j_1}, \ldots, s_x^{j_{|\mathcal{E}|}}\}$ of $|\mathcal{E}| \geq t+1$ partial function evaluation candidates originating from $|\mathcal{E}|$ different servers, and obtains verification keys $\mathsf{vk}_{j_1}, \ldots, \mathsf{vk}_{j_{|\mathcal{E}|}}$. Next,

1. Identifies an index subset $I = \{i_1, \ldots, i_{t+1}\}$ such that $\mathsf{VerifyEq}_{H_2}(g, \mathsf{vk}_i, H_1(x), v_i, \pi_i) =$ `accept` holds for every $i \in I$, where $(i, v_i, \pi_i) \leftarrow s_x^i$. If no such subset exists, outputs `reject`.
2. Sets $v \leftarrow \prod_{i \in I} v_i^{\lambda_{0,i,I}}$ and $\pi \leftarrow \{s_x^i\}_{i \in I}$.

3. Outputs (v, π).

$\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, x, v, \pi)$ parses $\pi = \{s_x^i\}_{i \in I}$ such that $|I| = t + 1$ and

1. Parses $(i, v_i, \pi_i) \leftarrow s_x^i$ for $i \in I$
2. Checks if $\mathsf{VerifyEq}_{H_2}(g, \mathsf{vk}_i, H_1(x), v_i, \pi_i) = \texttt{accept}$ for every $i \in I$; if some of the checks fail, outputs `reject`
3. Checks if $v = \prod_{i \in I} v_i^{\lambda_{0,i,I}}$; if so outputs `accept`; otherwise outputs `reject`.

Theorem 1. $\mathcal{V}^{\mathsf{DDH\text{-}DVRF}}$ *is admissible and correct.*

Proof (Sketch). The full proof can be found in Appendix B. First, we show that $\mathcal{V}^{\mathsf{DDH\text{-}DVRF}}$ is admissible. To see that consistency and domain range correctness are satisfied by $\mathcal{V}^{\mathsf{DDH\text{-}DVRF}}$ it suffices to see that the following equality holds:

$$\sum_{j \in \Delta} \mathsf{sk}_j \lambda_{0,j,\Delta} = \sum_{j \in \Delta} \lambda_{0,j,\Delta} \left(\sum_{i \in \mathrm{QUAL}} s_{i,j} \right) = \sum_{i \in \mathrm{QUAL}} \left(\sum_{j \in \mathcal{I}} \lambda_{0,j,\Delta} \cdot s_{i,j} \right)$$
$$= \sum_{i \in \mathrm{QUAL}} \left(\sum_{j \in \Delta} \lambda_{0,j,\Delta} \cdot f_i(j) \right) = \sum_{i \in \mathrm{QUAL}} a_{i,0} = \mathsf{sk}$$

Then $\prod_{j \in \Delta} (H_1(x)^{\mathsf{sk}_j})^{\lambda_{0,j,\Delta}} = H_1(x)^{\left(\sum_{j \in \Delta} \lambda_{0,j,\Delta} \cdot \mathsf{sk}_j\right)} = H_1(x)^{\mathsf{sk}}$ holds for every subset $\Delta \subseteq \{1, \ldots, \ell\}$ with $|\Delta| = t + 1$.

Domain range correctness is obvious and omitted. Provability follows from the completeness property of the NIZKs and the consistency of the DVRF. Uniqueness can be proven using the extractability of the NIZKs. That is, for any (v, π), if π verifies, we can extract a k such that $v = H_1(x)^k$ and $\mathsf{vk} = g^k$. The correctness is also proven using the extractability of the NIZKs. The NIZKs guarantee that the adversary is not able to include junk data in the computation of the pseudorandom function.

Theorem 2. $\mathcal{V}^{\mathsf{DDH\text{-}DVRF}}$ *is strongly pseudorandom under the* DDH *assumption in the random oracle model.*

Proof (Sketch). The full proof is given in Appendix B. The proof goes by constructing a sequence of hybrid games: $\mathsf{Hyb}_{\mathcal{A}}^{zk}(b)$, $\mathsf{Hyb}_{\mathcal{A}}^{sim}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$ for $0 \leq k \leq q_E$ where q_E is the total number of distinct x in the evaluation/challenge queries. This proof structure is similar to [AMMR18] but we are dealing with the distributed key generation protocol (the secure DKG protocol described in Figure 1) instead of a trusted dealer used in [AMMR18].

$\mathsf{Hyb}_{\mathcal{A}}^{zk}(b)$ replaces all the NIZKs in the original strong pseudorandomness game $\mathsf{PseudoRand}_{\mathcal{A}}(b)$ with simulated proofs using a random oracle H_2. $\mathsf{PseudoRand}_{\mathcal{A}}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{zk}(b)$ are indistinguishable due to the zero-knowledge property of the NIZKs.

$\mathsf{Hyb}_{\mathcal{A}}^{sim}(b)$ is defined exactly the same as $\mathsf{Hyb}_{\mathcal{A}}^{zk}(b)$ except a simulator [GJKR07] is used to simulate the DKG protocol. W.l.o.g., assume the participating servers are $S = \{1, \ldots, \ell\}$ and the adversary corrupts servers $C = \{1, \ldots, m\}$ with $m \leq t$. The simulator acts on behalf of all the honest servers $\{m + 1, \ldots, \ell\}$, while the adversary controls all the corrupted servers $\{1, \ldots, m\}$. The generating phase is run exactly the same as described in Figure 1, which determines a set of non-disqualified servers QUAL. The set QUAL contains all the honest servers and some of the corrupted servers. Since the simulator runs all the honest servers, it obtains all the shares from the corrupted servers

and can recover all the polynomials f_i with $i \in \mathrm{QUAL} \cap C$ chosen by the adversary. [3] Assume the combined polynomial after the generating phase is f. In the extracting phase, the simulator modifies the last polynomial, i.e., the ℓ-th polynomial $f^\star_\ell$, to set the value of the global public key $\mathsf{pk} = g^\alpha$ with α unknown. $f^\star_\ell$ is constructed implicitly to take the values: $\alpha - \sum_{j \in \mathrm{QUAL} \setminus \{\ell\}} f_j(0)$, $f_\ell(1), \ldots, f_\ell(m)$, $f_\ell(m+1), \ldots, f_\ell(t)$. Note that the shares $f_\ell(1), \ldots, f_\ell(m)$ are given to the adversary in the generating phase, thus they cannot be changed. We can compute $A_{i,j}, A^\star_{\ell,j}$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $0 \le j \le t$ as below:

- For $i \in \mathrm{QUAL} \setminus \{\ell\}$, the simulator has all the coefficients $a_{i,j}$ for $0 \le j \le t$ for polynomial f_i, thus $A_{i,j} = g^{a_{i,j}}$ can be easily computed.
- For the ℓ-th polynomial $f^\star_\ell$, the simulator computes $A^\star_{\ell,0} = g^{\alpha_0} \prod_{i \in \mathrm{QUAL} \setminus \{\ell\}} A^{-1}_{i,0}$ and $A^\star_{\ell,j} = (A^\star_{\ell,0})^{\delta_{0,j}} \prod_{i=1}^{t} g^{f_\ell(i) \cdot \delta_{i,j}}$ where $\delta_{i,j}$ is the coefficient of x^j in the lagrange basis polynomial $\lambda_{i,T}(x)$ with $T = \{0, 1, \ldots, t\}$.

Therefore, the final polynomial $f^\star$ takes the values $f^\star(0) = \alpha$, $f^\star(1) = f(1)$, $\ldots$, $f^\star(t) = f(t)$. The global public key $\mathsf{pk} = A^\star_{\ell,0} \prod_{i \in \mathrm{QUAL} \setminus \{\ell\}} A_{i,0} = g^\alpha$. $\mathsf{Hyb}^{sim}_{\mathcal{A}}(b)$ and $\mathsf{Hyb}^{zk}_{\mathcal{A}}(b)$ are indistinguishable due to the secrecy property of the secure DKG prtoocol.

$\mathsf{Hyb}^{k}_{\mathcal{A}}(b)$ for $0 \le k \le q_E$ is defined exactly the same as $\mathsf{Hyb}^{sim}_{\mathcal{A}}(b)$ except the first k distinct plaintext x in the evaluation/challenge queries are answered using random polynomials $f^1, f^2, \ldots, f^k$ which only matches on the adversary's shares, i.e., $f^i(1) = f^\star(1), \ldots, f^i(m) = f^\star(m)$ for $1 \le i \le k$.

The indistinguishability between $\mathsf{Hyb}^{k-1}_{\mathcal{A}}(b)$ and $\mathsf{Hyb}^{k}_{\mathcal{A}}(b)$ for $1 \le k \le q_E$ relies on the extended DDH problem [AMMR18] which can be easily derived from the original DDH problem. Formally, the extended DDH problem is of the form $(g^{\alpha_0}, g^{\alpha_1}, \ldots, g^{\alpha_w}, g^\beta, y_0, y_1, \ldots, y_w)$ where either $y_0 = g^{\alpha_0 \beta}, y_1 = g^{\alpha_1 \beta}, \ldots, y_w = g^{\alpha_w \beta}$ or $y_0 \xleftarrow{R} \mathbb{G}, y_1 \xleftarrow{R} \mathbb{G}, \ldots, y_w \xleftarrow{R} \mathbb{G}$. Suppose there exists a PPT adversary $\mathcal{A}$ that can distinguish between the hybrids $\mathsf{Hyb}^{k-1}_{\mathcal{A}}(b)$ and $\mathsf{Hyb}^{k}_{\mathcal{A}}(b)$ with non-negligible probability then we can construct a PPT adversary $\mathcal{B}$ to break the extended DDH assumption using $\mathcal{A}$ as a subroutine. In the proof, $\mathcal{B}$ first simulates the DKG protocol. Assume the combined polynomial after the generating phase is f. In the extracting phase, the last polynomial, i.e., the ℓ-th polynomial $f^\star_\ell$, is constructed to take the values: $\alpha_0 - \sum_{j \in \mathrm{QUAL} \setminus \{\ell\}} f_j(0)$, $f_\ell(1)$, $\ldots$, $f_\ell(m)$, $\alpha_{m+1} - \sum_{j \in \mathrm{QUAL} \setminus \{\ell\}} f_j(m+1)$, $\ldots$, $\alpha_t - \sum_{j \in \mathrm{QUAL} \setminus \{\ell\}} f_j(t)$ where $\alpha_0, \alpha_{m+1}, \ldots, \alpha_t$ are from the extended DDH problem which are not known to $\mathcal{B}$. We show how to compute $A_{i,j}, A^\star_{\ell,j}$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $0 \le j \le t$:

- For $i \in \mathrm{QUAL} \setminus \{\ell\}$, $\mathcal{B}$ has all the coefficients $a_{i,j}$ for $0 \le j \le t$ for polynomial f_i, thus $A_{i,j} = g^{a_{i,j}}$ can be easily computed.
- For the ℓ-th polynomial $f^\star_\ell$, $\mathcal{B}$ sets $A^\star_{\ell,0} = g^{\alpha_0} \prod_{i \in \mathrm{QUAL} \setminus \{\ell\}} A^{-1}_{i,0}$ and $A^\star_{\ell,j} = (A^\star_{\ell,0})^{\delta_{0,j}} \prod_{i=1}^{m} g^{f_\ell(i) \cdot \delta_{i,j}}$ $\prod_{i=m+1}^{t} (g^{\alpha_i} \prod_{k \in \mathrm{QUAL} \setminus \{\ell\}} g^{-f_k(i)})^{\delta_{i,j}}$ where $\delta_{i,j}$ is the coefficient of x^j in the lagrange basis polynomial $\lambda_{i,T}(x)$ with $T = \{0, 1, \ldots, t\}$.

Let the final combined polynomial be $f^\star$. We know that $f^\star(0) = \alpha_0$, $f^\star(1) = f(1), \ldots, f^\star(m) = f(m)$, $f^\star(m+1) = \alpha_{m+1}, \ldots, f^\star(t) = \alpha_t$. This sets the global public key $\mathsf{pk}_0 = A^\star_{\ell,0} \prod_{i \in \mathrm{QUAL} \setminus \{\ell\}} A_{i,0}$ $= g^{\alpha_0}$ and the implicit secret value is $\mathsf{sk}_0 = \alpha_0$. The secret key sk_i for an honest server $m+1 \le i \le t$ is set to be $\mathsf{sk}_i = \alpha_i$ while the corresponding verification key is $\mathsf{vk}_i = g^{\alpha_i}$. The secret key sk_i for an honest server $i \ge t+1$ is set to be $\mathsf{sk}_i = \prod_{j \in T} \mathsf{sk}_j^{\lambda_{i,j,T}}$ where $T = \{0, 1, \ldots, t\}$ and $\lambda_{i,j,T}$ are

[3] This typically requires the condition $t < \ell/2$ which guarantees the honest parties can run the protocol without the corrupted parties, i.e., $\ell - t > t$ under the assumption that the adversary can compromise up to t servers. However, the condition $t < \ell/2$ is not a necessary condition. A more general description is $\ell_{\mathsf{hon}} > t \ge \ell_{\mathsf{corr}}$ where ℓ_{hon} is the number of the honest parties and ℓ_{corr} is the number of the corrupted servers.

the lagrange coeffients, while the corresponding verification key is $\mathsf{vk}_i = \prod_{j \in T} \mathsf{vk}_j^{\lambda_{i,j,T}}$. Note that, for each honest server i ($m+1 \leq i \leq \ell$), its secret key sk_i cannot be computed by $\mathcal{B}$ because α_js are unknown but the corresponding verification key vk_i can be computed. $\mathcal{B}$ chooses a random $\eta^\star \xleftarrow{R} [q_{H_1}]$ where q_{H_1} is the total number of distinct random oracle queries to H_1. For the $\eta^\star$-th query $x_{\eta^\star}$ to H_1, $\mathcal{B}$ sets $H_1(x_{\eta^\star}) = g^\beta$.

We shall briefly describe how to answer the evaluation queries. $\mathcal{B}$ chooses $k-1$ random polynomials f^i ($1 \leq i \leq k-1$) such that $f^i(1) = f^\star(1), \ldots, f^i(m) = f^\star(m)$. On the i-th distinct evaluation query (Eval, x, j) with $i < k$, $\mathcal{B}$ returns $(j, H_1(x)^{f^i(j)}, \pi_j)$ where π_j is a simulated proof generated using random oracle H_2. On the k-th distinct evaluation query on (Eval, x, j), $\mathcal{B}$ answers with (j, y_j, π_j) for $m+1 \leq j \leq t$, while answers with $(\prod_{i \in T} y_i^{\lambda_{j,i,T}}, \pi_j)$ for $j \geq t$, where y_is are from the DDH problem and π_j is a simulated proof. On the i-th distinct evaluation query (Eval, x, j) with $i > k$, $\mathcal{B}$ returns $(j, \mathsf{vk}_j^r, \pi_j)$ with π_j a simulated proof. With $1/q_{H_1}$ probability (q_{H_1} is polynomially bounded in the security parameter), the $x_{\eta^\star}$ is also the k-th distinct evaluation/challenge query x_k. Let's assume this is the case. When $y_i = g^{\alpha_i \beta}$, $\mathcal{B}$ answers the evaluation queries on x_k correctly, thus it simulates $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ perfectly; when $y_i \xleftarrow{R} \mathbb{G}$, the partial evaluation/challenge queries on x_k are answered using random polynomials, thus it simulates $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$. Therefore, $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$ are indistinguishable under DDH assumption.

Finally $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(0)$ and $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(1)$ can be shown to be indistinguishable because all the evaluation/challenge queries are answered using a unique random polynomial. This concludes the proof of the indistinguishablity of $\mathsf{PseudoRand}_{\mathcal{A}}(0)$ and $\mathsf{PseudoRand}_{\mathcal{A}}(1)$.

5 Pairings-based DVRF with compact proofs

The construction of the non-interactive DDH-based DVRF is simple and can be implemented efficiently on any elliptic curve DDH group, but it cannot provide compact proofs π for verifying the pseudorandom value v, i.e., π is linear in the threshold t. Constructing a DVRF based on cryptographic pairings can achieve compact proofs, probably at the expense of computational efficiency, due to the slightly less efficient operations on pairing groups.

In this section, we shall first formalize one such a pairing-based construction that was sketched by Hanke, Movahedi and Williams [HMW18] and that we call Dfinity-DVRF. We notice that no formal treatment nor security proofs have been given to our knowledge. We show that Dfinity-DVRF *achieves standard pseudorandomness*. Next, we shall propose a new pairing-based non-interactive DVRF, called GLOW-DVRF, that has *strong pseudorandomness and compact proofs*, while also improving on the efficiency of Dfinity-DVRF.

5.1 Formalisation of Dfinity-DVRF

As outlined in [HMW18], Dfinity-DVRF uses a well-known DKG protocol by Pedersen [Ped91] where the adversary can bias the distribution of the joint public key [GJKR07]. In order to formalise Dfinity-DVRF and study its security properties, instead of the biasable DKG protocol we use the secure DKG protocol proposed in [GJKR07] and we shall prove that the resulting DVRF construction satisfies standard pseudorandomness.

Let $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2, h_1, h_2)$ be a bilinear pairing group where the co-CDH assumption holds and g_1, g_2 are generators of $\mathbb{G}_1, \mathbb{G}_2$ respectively (same applies to h_1, h_2). Let $H_1 : \{0,1\}^* \to \mathbb{G}_1$, $H_2 : \mathbb{G}_1 \to \{0,1\}^{b(\lambda)}$ be hash functions. $\mathcal{V}^{\mathsf{Dfinity\text{-}DVRF}}$ is defined as follows:

$\mathsf{DistKG}(1^\lambda, t, \ell)$ proceeds in the same way as in the non-compact case, with the difference that exponentiations take place in group $\mathbb{G}_2$, i.e. the verification keys $\mathsf{vk}_i = g_2^{\mathsf{sk}_i} \in \mathbb{G}_2$ and the global public key $\mathsf{pk} = \sum_{i \in \mathrm{QUAL}} g_2^{a_{i,0}} \in \mathbb{G}_2$. Full details can be found in Figure 2 in Appendix C.

$\mathsf{PartialEval}(\mathsf{sk}_i, x)$ outputs a share $s_x^i = (i, v_i)$ with $v_i = H_1(x)^{\mathsf{sk}_i}$.

$\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, x, \mathcal{E})$: parses list $\mathcal{E} = \{s_x^{j_1}, \ldots, s_x^{j_{|\mathcal{E}|}}\}$ of $|\mathcal{E}| \geq t+1$ partial evaluation candidates originating from $|\mathcal{E}|$ different servers, and obtains verification keys $\mathsf{vk}_{j_1}, \ldots, \mathsf{vk}_{j_{|\mathcal{E}|}}$. Next,

1. Identifies an index subset $I = \{i_1, \ldots, i_{t+1}\}$ such that for every $i \in I$ it holds that $e(v_i, g_2) = e(H_1(x), \mathsf{vk}_i)$. If no such subset exists, outputs `reject`.
2. Sets $\pi \leftarrow \prod_{j \in I} v_j^{\lambda_{0,j,I}}$ and $v = H_2(\pi)$.
3. Outputs (v, π).

$\mathsf{Verify}(\mathsf{pk}, x, (v, \pi))$ outputs `accept` if, given a public key pk and a plaintext x, the relation holds: $e(\pi, g_2) = e(H_1(x), \mathsf{pk})$ and $v = H_2(\pi)$. Otherwise output `reject`.

Theorem 3. $\mathcal{V}^{\mathsf{Dfinity\text{-}DVRF}}$ *is admissible and correct.*

Proof. For admissibility: the proof of consistency and domain range correctness is essentially the same as in Theorem 1. The provability follows from the fact that if $e(v_i, g_2) = e(H_1(x), \mathsf{vk}_i)$ then we can derive $v_i = H_1(x)^{\mathsf{sk}_i}$. Similarly, the uniqueness follows from the fact that if $e(v, g_2) = e(H_1(x), \mathsf{pk}) = e(v', g_2)$ then $v = v' = H_1(x)^{\mathsf{sk}}$.

For correctness: the proof is similar to that of Theorem 1, except that it does not need to use the random oracle model and the NIZKs are replaced with pairing equations. Suppose there exists an adversary $\mathcal{A}$ leading the challenger to output 1 with non-negligible probability, i.e., $v^\star \neq v$. This means there exists $i \in U$ such that $v_i^\star \neq v_i$. If $i \in U \setminus C$, this is not possible since $v_i^\star = v_i = H_1(x^\star)^{\mathsf{sk}_i}$. If $i \in U \cap C$, we know that $e(v_i^\star, g_2) = e(H_1(x^\star), \mathsf{vk}_i)$, where $\{(i, v_i^\star)\}_{i \in U \cap C}$ and $U \subseteq \mathrm{QUAL}$ are evaluation shares received from the adversary. This gives us that $v_i^\star = H_1(x^\star)^{\mathsf{sk}_i} = v_i$ for $i \in U \cap C$ which contradicts the hypothesis.

Theorem 4. $\mathcal{V}^{\mathsf{Dfinity\text{-}DVRF}}$ *achieves standard pseudorandomness under the co-CDH assumption.*

Proof (Sketch). The full proof is given in Appendix C. Recall that standard pseudorandomness does not allow the adversary to query partial evaluations on the challenge plaintext $x^\star$. This implies that standard pseudorandomness can be derived from the unforgeability of the threshold BLS signature [Bol02]. The definition of unforgeability of the threshold BLS does not allow an adversary to obtain any partial signature on the challenge plaintext even if the adversary compromises less than t servers. Similarly to Theorem 2, we construct the hybrid $\mathsf{Hyb}^{sim}(b)$ to simulate the DKG protocol, and show that $\mathsf{Hyb}^{sim}(0)$ and $\mathsf{Hyb}^{sim}(1)$ are indistinguishable under the co-CDH assumption.

Suppose there exists an adversary $\mathcal{A}$ that distinguishes $\mathsf{Hyb}^{sim}(0)$ and $\mathsf{Hyb}^{sim}(1)$, then we can construct an adversary $\mathcal{B}$ that breaks the co-CDH assumption using $\mathcal{A}$ as a subroutine. Given a co-CDH instance 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 $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, $\alpha, \beta \xleftarrow{R} \mathbb{Z}_q$, adversary $\mathcal{B}$'s goal is to output $g_1^{\alpha\beta}$. In the DKG setup, $\mathcal{B}$ simulates the DKG protocol to interact with the corrupted servers. The protocol constructs $C_{i,k}$ using g_2, h_2 from $\mathbb{G}_2$. The computation of $A_{i,j}, A_{\ell,j}^\star$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $0 \leq j \leq t$ are similar to $\mathsf{Hyb}_{\mathcal{A}}^{sim}(b)$ in Theorem 2 except they are constructed on $\mathbb{G}_2$. The global public key is $\mathsf{pk} = g_2^\alpha$. Similarly, the simulator can compute $(\mathsf{sk}_i, \mathsf{vk}_i)$ for honest servers $m+1 \leq i \leq t$, while it cannot compute vk_i but not sk_i for $i \geq t+1$ because α is unknown. $\mathcal{B}$ chooses a random $\eta^\star \xleftarrow{R} [q_{H_1}]$ where q_{H_1} is the total number of distinct random oracle queries to H_1. For the $\eta^\star$-th query $x_{\eta^\star}$ to H_1, $\mathcal{B}$ sets $H_1(x_{\eta^\star}) = g_1^\beta$. With $1/q_{H_1}$ probability (q_{H_1} is polynomial), $x_{\eta^\star}$ is also the challenge query $x^\star$. Let's assume this is the case. Because the adversary is not allowed to query partial evaluations on the challenge plaintext, we don't need to handle the evaluation query on $x_{\eta^\star}$. The other evaluation queries (Eval, x, j) with $x \neq x^\star$ can be easily answered by computing vk_j^r where $(x, r, g^r) \in \mathcal{L}_{H_1}$ is an entry in the random oracle H_1 and r is chosen randomly by $\mathcal{B}$. On the challenge query, the simulator checks if there exists any entry $(y, c) \in \mathcal{L}_{H_2}$ in the random oracle H_2 such that $e(y, g_2) = e(g_1^\beta, g_2^\alpha)$ which means $y = g_1^{\alpha\beta}$. If not,

$\mathcal{B}$ chooses a random c and adds the entry $(\perp, c)$ to the random oracle H_2. $\mathcal{B}$ returns c if $b = 0$ and returns a random if $b = 1$.

We define an event C to hold when the adversary queries $y^\star$ to H_2 such that $e(y^\star, g_2) = e(g_1^\beta, g_2^\alpha)$, i.e., $y^\star = g_1^{\alpha\beta}$. Naturally, when C happens, $\mathcal{B}$ can output $y^\star$ as a solution to the co-CDH problem. Now we shall show that the probability $\Pr[C]$ that C happens is non-negligible. We can prove that the probability that C does not happen is the same for $b = 0$ and $b = 1$, i.e., $\Pr[\neg C|b = 0] = \Pr[\neg C|b = 1]$ due to the fact that the adversary sees the same probability distribution on $\mathcal{B}$'s outputs before querying $y^\star$ to H_2 for the first time. Based on this, we can also derive that $\Pr[C|b = 0] = \Pr[C|b = 1]$ and $\Pr[C] = \Pr[C|b = 0]\Pr[b = 0] + \Pr[C|b = 1]\Pr[b = 1] = \Pr[C|b = 0]$. Obviously, when C does not happen, the adversary sees the same probability distribution on $\mathcal{B}$'s outputs, and thus the probability that $\mathcal{A}$ outputs 1 is also the same for $b = 0$ and $b = 1$, i.e., $\Pr[b' = 1|b = 0, \neg C] = \Pr[b' = 1|b = 1, \neg C]$. Therefore we can compute $\mathcal{A}$'s advantage of distinguishing $b = 0$ and $b = 1$ as

$$
\begin{aligned}
\mathbf{Adv} &= |\Pr[b' = 1|b = 0] - \Pr[b' = 1|b = 1]| \\
&= |\Pr[b' = 1|b = 0, C]\Pr[C|b = 0] + (\Pr[b' = 1|b = 0, \neg C]\Pr[\neg C|b = 0] \\
&\quad - \Pr[b' = 1|b = 1, C]\Pr[C|b = 1] - \Pr[b' = 1|b = 1, \neg C]\Pr[\neg C|b = 1]| \\
&= |\Pr[b' = 1|b = 0, C]\Pr[C|b = 0] - \Pr[b' = 1|b = 1, C]\Pr[C|b = 1]| \\
&= |\Pr[b' = 1|b = 0, C] - \Pr[b' = 1|b = 1, C]| \cdot \Pr[C|b = 0] \\
&\leq \Pr[C|b = 0] = \Pr[C]
\end{aligned}
$$

Since we assume $\mathcal{A}$ distinguishes $b = 0$ and $b = 1$ with non-negligible advantage $\mathbf{Adv}$, C happens with non-negligible probability $\Pr[C]$. Therefore $\mathcal{B}$ outputs y as a solution to the co-CDH problem with non-negligible probability $\Pr[C]/q_{H_1}$.

Strong pseudorandomness of Dfinity-DVRF. Adversarial capabilities in standard pseudorandomness of Dfinity-DVRF and unforgeability of threshold BLS signatures [Bol02] go hand in hand, as in both cases an adversary is not given the option to make partial queries on the challenging plaintext $x^\star$. It is not surprising then that the previous security reduction does not allow to prove strong pseudorandomness of Dfinity-DVRF. Indeed, since the verification key vk_i of the i-th server is in G_2, then the adversary can verify if an answer v_i is correct or not by checking the pairing $e(v_i, g_2) = e(H_1(x^\star), \mathsf{vk}_i)$. Alas, the security reduction does not provide the challenger with knowledge of $\log_{g_2} \mathsf{vk}_i$, and then the challenger cannot answer a partial signature/evaluation query on the challenge plaintext.

Next we propose a new pairing-based DVRF called GLOW-DVRF that has strong pseudorandomness and improved running times with respect to Dfinity-DVRF.

5.2 An improved pairing-based DVRF: GLOW-DVRF

In order to achieve and prove strong pseudorandomness, we need to perform a few modifications to Dfinity-DVRF, most notably:

- The DKG protocol generates the verification keys $\mathsf{vk}_i = g_1^{\mathsf{sk}_i}$ in $\mathbb{G}_1$ and the global public key $\mathsf{pk} = \prod_{i \in \mathrm{QUAL}} g_2^{a_{i,0}}$ in $\mathbb{G}_2$.
- The $\mathsf{PartialEval}(\mathsf{sk}_i, x)$ algorithm produces a NIZK to show the partial evaluation is correctly formed.

In comparison to Dfinity-DVRF, having the i-th verification key $\mathsf{vk}_i \in \mathbb{G}_1$, does not allow an adversary to check the validity of the i-th partial evaluation (provided that the eXternal DDH assumption holds in $\mathbb{G}_1$). Instead, the well-formedness of $\mathsf{PartialEval}(\mathsf{sk}_i, \mathsf{vk}_i, x)$ will be checked by verifying a NIZK proof of equality of discrete logarithms. This allows the security reduction to simulate $\mathsf{PartialEval}(\mathsf{sk}_i, \mathsf{vk}_i, x^\star)$, where $x^\star$ is the challenge plaintext.

Let $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2, h_1, h_2)$ be a bilinear pairing where the DDH assumption holds in $\mathbb{G}_1$, where g_1, g_2 are generators of $\mathbb{G}_1, \mathbb{G}_2$ respectively (same applies to h_1, h_2). Let $H_1 : \{0,1\}^* \to \mathbb{G}_1$, $H_2 : \mathbb{G}_1 \to \{0,1\}^{b(k)}$ and $H_3 : \{0,1\}^* \to \mathbb{Z}_q$ be hash functions. Let $\mathcal{V}^{\mathsf{GLOW\text{-}DVRF}} = (\mathsf{DistKG}, \mathsf{PartialEval}, \mathsf{Combine}, \mathsf{Verify})$ be a DVRF for a pseudo-random function $F_{\mathsf{sk}} : \{0,1\}^{a(\lambda)} \to \{0,1\}^{b(\lambda)}$ that is defined as follows:

$\mathsf{DistKG}(1^\lambda, t, \ell)$ proceeds in the same way as in the non-compact case in Figure 1, with the difference described as below:

- In the Generating phase, g, h are replaced with $g_1, h_1 \in \mathbb{G}_1$.
- In Step 4, each server S_i with $i \in \text{QUAL}$ exposes $B_{i,0} = g_2^{a_{i,0}}$ via Feldman-VSS.
  - In Step 4(a), each server S_i with $i \in \text{QUAL}$ broadcasts $A_{i,k} = g_1^{a_{i,k}}$ for $0 \le k \le t$ and $B_{i,0} = g_2^{a_{i,0}}$.
  - In Step 4(b), for each $i \in \text{QUAL}$, S_j checks if $g_1^{s_{i,j}} = \prod_{k=0}^{t} (A_{i,k})^{j^k}$ and $e(A_{i,0}, g_2) = e(g_1, B_{i,0})$.
  - In Step 4(c), set the global public key as $\mathsf{pk} = \prod_{i \in \text{QUAL}} B_{i,0}$.

  To summarise, the verification keys vk_i are generated on $\mathbb{G}_1$ using the generator g_1 and the global public key pk are generated on $\mathbb{G}_2$ using the generator g_2. Full details can be found in Figure 3 in Appendix D.

$\mathsf{PartialEval}(\mathsf{sk}_i, x)$ outputs a share $s_x^i = (i, v_i, \pi_i)$ with $v_i = H_1(x)^{\mathsf{sk}_i}$ and $\pi_i \leftarrow \mathsf{PrEq}_{H_3}(g, \mathsf{vk}_i, H_1(x), v_i; r)$ for randomness $r \xleftarrow{R} \mathbb{Z}_q$.

$\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, x, \mathcal{E})$ parses list $\mathcal{E} = \{s_x^{j_1}, \ldots, s_x^{j_{|\mathcal{E}|}}\}$ of $|\mathcal{E}| \ge t+1$ partial evaluation candidates originating from $|\mathcal{E}|$ different servers, and obtain verification keys $\mathsf{vk}_{j_1}, \ldots, \mathsf{vk}_{j_{|\mathcal{E}|}}$. Next,

1. Identifies an index subset $I = \{i_1, \ldots, i_{t+1}\}$ such that for every $i \in I$ it holds that $\mathsf{VerifyEq}_{H_3}(g, \mathsf{vk}_i, H_1(x), v_i, \pi_i) = \texttt{accept}$, where $(i, v_i, \pi_i) \leftarrow s_x^i$. If no such subset exists, outputs `reject`.
2. Sets $\pi \leftarrow \prod_{j \in I} v_j^{\lambda_{0,j,I}}$ and $v = H_2(\pi)$.
3. Outputs (v, π).

$\mathsf{Verify}(\mathsf{pk}, x, v, \pi)$**:** output `accept` if given a public key pk and a plaintext x, the relation holds: $e(\pi, g_2) = e(H_1(x), \mathsf{pk})$ and $v = H_2(\pi)$. Otherwise output `reject`.

Note that while in GLOW-DVRF partial evaluations are validated by verifying NIZKs, the well-formedness of the pseudorandom value v is still validated using a pairing equation, which provides the compact proof for the pseudorandom value as in Dfinity-DVRF. Using NIZKs to validate partial evaluations can speed up the Combine algorithm by computing $4 \cdot |\mathcal{E}|$ exponentiations in $\mathbb{G}_1$ instead of computing $2 \cdot |\mathcal{E}|$ pairings. This potentially makes our scheme x2.5 to x4.5 faster than Dfinity-DVRF, depending on the ratio of computing an exponentiation in $\mathbb{G}_1$ versus computing a pairing (see Section 6.2 for concrete experimental values).

One may argue that Dfinity-DVRF allows the Combine algorithm to verify partial evaluations in batches in order to reduce the number of pairings to be computed. However, batch verification is ill-suited for the Combine algorithm for three reasons: i) batch verification is meaningful only when every individual verification is successful; when verification fails, the partial evaluations would still need to be verified one-by-one to find the ill-formed input; ii) batch verification requires additional randomisation before coalescing the pairings (e.g., [BDN18]). That is, for each partial evaluation share v_i, it needs to choose a random r_i and compute $v_i^{r_i}$ and $\mathsf{vk}_i^{r_i}$. This introduces additional $|\mathcal{E}|$ exponentiations on $\mathbb{G}_1$ and $|\mathcal{E}|$ exponentiations on $\mathbb{G}_2$. The exponentiation in $\mathbb{G}_2$ is about 2 times slower than in $\mathbb{G}_1$, e.g., [Mis19]; iii) using batch verification here is not better than applying the lagrange coefficients to directly compute the pseudorandom function v without verifying the partial evaluations and then checking if the result is correct by testing if $e(v, g_2) = e(H_1(x), \mathsf{pk})$. This can be performed in our scheme as well.

Theorem 5. $\mathcal{V}^{\mathsf{GLOW\text{-}DVRF}}$ *is admissible and correct.*

The proof of admissibility and correctness is similar to Theorem 1 and is omitted.

Theorem 6. $\mathcal{V}^{\mathsf{GLOW\text{-}DVRF}}$ *is strongly pseudorandom under the* XDH *assumption and* co-CDH *assumption in the random oracle model.*

Proof (Sketch). The full proof is given in Appendix D. Similar to Theorem 2, we construct $\mathsf{Hyb}^{zk}(b)$ to simulate all the NIZKs using random oracle H_3 and construct $\mathsf{Hyb}^{sim}(b)$ to simulate the DKG protocol. We have that the original strong pseudorandomness game and $\mathsf{Hyb}^{zk}(b)$ are indistinguishable due to the zero-knowledge property of NIZKs, and $\mathsf{Hyb}^{zk}(b)$ and $\mathsf{Hyb}^{sim}(b)$ are indistinguishable due to the secrecy of the simulator. Assume m is the total number of corrupted servers, we consider two cases $m = t$ and $m < t$.

The case when $m = t$**:** this is the simplest case and the proof is similar to Theorem 4. This is because $m = t$ means an adversary has already compromised t servers and cannot issue any evaluation query on the challenge plaintext $x^\star$. Given an adversary $\mathcal{A}$ that distinguishes $\mathsf{Hyb}^{sim}(0)$ and $\mathsf{Hyb}^{sim}(1)$, the construction of an adversary $\mathcal{B}$ breaks the co-CDH assumption using $\mathcal{A}$ as a subroutine is exactly the same as the proof of Theorem 4 except that

1. In the DKG setup, the verification keys are generated in $\mathbb{G}_1$ and the global public key is generated in $\mathbb{G}_2$.
2. In the challenge query, the challenger verifies the NIZKs to identify a set of correct partial evaluation shares.
3. An additional random oracle H_3 is constructed in the standard way.

The rest of the analysis is similar to Theorem 4: we can prove that $\mathcal{B}$ outputs a solution to the co-CDH problem with non-negligible probability $\Pr[C]/q_{H_1}$.

The case when $m < t$**:** in this case, the adversary is allowed to make up to $t - m(\geq 1)$ evaluation queries on the challenge plaintext $x^\star$. To prove the strong pseudorandomness, we shall construct k-hybrids $\mathsf{Hyb}^k_{\mathcal{A}}(b)$ with $0 \leq k \leq q_E$ and q_E the total number of distinct x in the evaluation/challenge queries. We prove that

1. $\mathsf{Hyb}^{k-1}_{\mathcal{A}}(b)$ and $\mathsf{Hyb}^k_{\mathcal{A}}(b)$ for $1 \leq k \leq q_E$ are indistinguishable under XDH assumption. This part of the proof is similar to Theorem 2 except that the global public key $\mathsf{pk} = g_2^\gamma$ where $\gamma \xleftarrow{R} \mathbb{Z}_q$ is chosen by the simulator and cannot be changed throughout the game. Assume the final polynomial after the DKG protocol is $f^\star$. When the simulator chooses random polynomials $f^1, f^2, \ldots, f^k$ to answer the first k distinct evaluation/challenge queries, these polynomials need to satisfy $f^i(0) = \gamma$ in addition to $f^i(1) = f^\star(1), \ldots, f^i(m) = f^\star(m)$. This is because the global public key is constructed on $\mathbb{G}_2$. If the simulator changes the value of $f^i(0)$, the adversary can detect it by issuing $t + 1$ queries on a plaintext $x(\neq x^\star)$ and then recovering the pseudorandom value v and checking if $e(v, g_2) = e(H_1(x), \mathsf{pk})$. This is not a problem for Theorem 2 because the global public key is generated on $\mathbb{G}_1$ and the NIZKs can be simulated.
2. $\mathsf{Hyb}^{q_E}_{\mathcal{A}}(0)$ and $\mathsf{Hyb}^{q_E}_{\mathcal{A}}(1)$ are indistinguishable under co-CDH assumption. Suppose there exists an adversary $\mathcal{A}$ that distinguishes $\mathsf{Hyb}^{q_E}_{\mathcal{A}}(0)$ and $\mathsf{Hyb}^{q_E}_{\mathcal{A}}(1)$, 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 $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, $\alpha, \beta \xleftarrow{R} \mathbb{Z}_q$. $\mathcal{B}$'s goal is to output $g_1^{\alpha\beta}$. $\mathcal{B}$ simulates the DKG protocol to interact with the corrupted servers. Let QUAL be the non-disqualified servers determined after the generating phase. Assume the combined polynomial after the generating phase is f. In the extracting phase, for the ℓ-th polynomial, $\mathcal{B}$ constructs $f^\star_\ell$ to replace f_ℓ: $f^\star_\ell$ has the values $\alpha - \sum_{j \in \mathrm{QUAL} \setminus \{\ell\}} f_j(0)$, $f_\ell(1)$, $\ldots$, $f_\ell(t)$. $A_{i,j}, A^\star_{\ell,j}$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $0 \leq j \leq t$ can be computed similar to [GJKR07]. $B^\star_{i,0}$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $B^\star_{\ell,0}$ are computed as:

– For $i \in \mathrm{QUAL} \setminus \{\ell\}$, $B_{i,0} = g_2^{a_{i,0}}$ since $\mathcal{B}$ has the coefficients of all the polynomials except the ℓ-th polynomial.
– $B^\star_{\ell,0} = g_2^\alpha \prod_{i \in \mathrm{QUAL} \setminus \{\ell\}} B_{i,0}^{-1}$.

The global public key is $\mathsf{pk} = B^\star_{\ell,0} \prod_{i \in \mathrm{QUAL} \setminus \{\ell\}} B_{i,0} = g_2^\alpha$. Let the final combined polynomial be $f^\star$. We know that $f^\star(0) = \alpha$, $f^\star(1) = f(1), \ldots, f^\star(t) = f(t)$. The corrupted servers that are included in QUAL have the shares $\mathsf{sk}_i = f^\star(i)$ and $\mathsf{vk}_i = g_1^{f^\star(i)}$ for $i \in \mathrm{QUAL} \cap C$. The shares of the honest servers are set to be $\mathsf{sk}_i = f^\star(i)$ and $\mathsf{vk}_i = g_1^{f^\star(i)}$ for $m+1 \leq i \leq \ell$. Note that, for the honest servers $t+1 \leq i \leq \ell$, the simulator cannot compute sk_i because it does not know the value of α, but can compute $\mathsf{vk}_i = g_1^{\alpha \lambda_{i,0,T}} \prod_{j \in T, j \neq 0} g_1^{f^\star(j)\lambda_{i,j,T}}$ with $T = \{0, 1, \ldots, t\}$. $\mathcal{B}$ chooses a random $\eta^\star \xleftarrow{R} [q_{H_1}]$ where q_{H_1} is the total number of distinct random oracle queries to H_1. For the $\eta^\star$-th query $x_{\eta^\star}$ to H_1, $\mathcal{B}$ sets $H_1(x_{\eta^\star}) = g_1^\beta$. With $1/q_{H_1}$ probability (q_{H_1} is polynomial), $x_{\eta^\star}$ is also the challenge query. Let's assume this is the case from now on. $\mathcal{B}$ chooses q_E random polynomials $f^1, f^2, \ldots, f^{q_E}$ such that for $1 \leq j \leq q_E$, $f^j(0) = \alpha$ and for all $i \in C$, $f^j(i) = f^\star(i)$. Note that, we cannot compute $f^j(i)$ for $i \geq m+1$ because α is unknown, but we can compute $g_1^{f^j(i)}$ in the way similar to vk_i. For the i-the distinct evaluation query $x (\neq x^\star)$, $\mathcal{B}$ answers with $(g_1^{f^j(i) \cdot r}, \pi_i)$ where (x, r, g_1^r) is an entry in the random oracle H_1 and π_i is a simulated proof. For each evaluation query (Eval, x, i) on the challenge plaintext $x^\star$, $\mathcal{B}$ answers with a random z_i. The adversary is allowed to make at most $t - m$ evaluation queries on $x^\star$. Let these evaluation queries be $(\mathsf{Eval}, x^\star, i_1), \ldots, (\mathsf{Eval}, x^\star, i_{t-m})$ and the returned randoms be $z_{i_1}, \ldots, z_{i_{t-m}}$. These values $(i_1, z_{i_1}), \ldots, (i_{t-m}, z_{i_{t-m}})$ together with $(0, \alpha\beta)$ and $(1, f^\star(1)\beta), \ldots, (m, f^\star(m)\beta)$ implicitly defines an unique random polynomial $\hat{f}$ such that $\hat{f}(0) = \alpha$, $\hat{f}(1) = f^\star(1)$, $\ldots$, $\hat{f}(m) = f^\star(m)$, and $\hat{f}$ is only used for computing evaluations on $x^\star$. Note that, it does not matter if the adversary makes less than $t - m$ evaluation queries on $x^\star$ since the polynomial $\hat{f}$ will never be explicitly computed. Therefore, $\mathcal{B}$ simulates $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(b)$ perfectly.

The rest of the analysis is similar to the one in the case $m = t$. We define an event C as when the adversary queries $y^\star$ to H_2 such that $e(y^\star, g_2) = e(g_1^\beta, g_2^\alpha)$, i.e., $y^\star = g_1^{\alpha\beta}$. We can prove that $\mathcal{A}$'s advantage $\mathbf{Adv}$ of distinguishing $b = 0$ and $b = 1$ is not bigger than $\Pr[C]$, i.e., $\mathbf{Adv} \leq \Pr[C]$. Since we assume $\mathcal{A}$ distinguishes $b = 0$ and $b = 1$ with non-negligible advantage $\mathbf{Adv}$, C happens with non-negligible probability $\Pr[C]$. Therefore $\mathcal{B}$ outputs $y^\star$ as a solution to the co-CDH problem with non-negligible probability $\Pr[C]/q_{H_1}$.

6 Decentralised Random Beacon and Performance Evaluation

A Decentralised Random Beacon (DRB) provides a way to distributedly agree on a randomly chosen leader in Proof-of-Stake blockchains (e.g. Dfinity [HMW18], Ethereum 2.0 [But18], OmniLedger [KJG+18], Tendermint [BKM18]), without the need for a coordinator. DRBs are a particular case of Multi-Party Computation protocol and it can be straightforwardly obtained from a DVRF as follows.

The DRB is initiated during a Setup phase that involves a set of ℓ nodes that can participate in the consensus and in essence it corresponds to the DVRF Setup phase. At the end of the Setup phase, each node that followed protocols ends up with a pair of verification and secret keys $(\mathsf{vk}_i, \mathsf{sk}_i)$ for the i-th node, as well as a global key pair $(\mathsf{pk}, \mathsf{sk})$. The latter key pair implicitly defines the VRF function $F_{\mathsf{sk}}(\cdot)\cdot$, where sk can be thought of as a virtual secret key that is never computed explicitly. The set of nodes' secret keys thereby computed enable any subset of nodes of size $t+1$ to compute the verifiable random value $F_{\mathsf{sk}}(x)$ for a publicly known input x, via running $\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, x, \mathcal{E})$ for a list $\mathcal{E} = \{s_x^{j_1}, \ldots, s_x^{j_{|\mathcal{E}|}}\}$ of $|\mathcal{E}| \geq t+1$ partial function evaluation candidates originating from $|\mathcal{E}|$ different servers. Conversely, any set of at most h nodes cannot learn any information on $F_{\mathsf{sk}}(x)$ for

any x not previously evaluated. It is assumed that an adversary that wants to predict the random value for future rounds will not control more than h nodes at any point in time, where typically $h = \ell/3$.

Successive random values σ_r for a given round $r \geq 1$ are defined as $\sigma_r \leftarrow F_{\mathsf{sk}}(\sigma_{r-1}||r)$, starting from a publicly known initial seed σ_0, not controlled by an adversary. The output σ_r of the VRF can be verified against its proof of correctness $\pi_{(\sigma_{r-1}||r)}$, by running the verification algorithm of the concrete DVRF protocol by running

$$\mathsf{Verify}(\mathsf{pk}, x, \sigma_r, \pi_x) = \begin{cases} 1 \text{ if true} \\ 0 \text{ otherwise} \end{cases} \quad \text{where } \; x = \sigma_{r-1}||r$$

Choosing the initial seed. The seed initial σ_0 to bootstrap the DRB should be free from manipulation by an attacker. Previous DRB constructions have not convincingly provided an explicit initial seed free from adversarial bias. We propose as explicit initial seed $\sigma_0 := \mathsf{pk}$, where pk is the global public key computed in any of our concrete constructions. Indeed pk is uniformly distributed at random in the corresponding DH group, a property inherited by our DKG protocols from that of Gennaro *et al.* [GJKR07].

Strong bias resistance. The uniqueness of the pseudorandom output of our constructions provides the strongest form of bias resistance, as it stands against any adversary independently from the number of corrupted servers that the adversary controls.

6.1 Theoretical performance

In Table 1, we give analytical measurements for the size of the output of the partial evaluation and the combine algorithms of the DVRFs studied in this paper. The size is represented using the number of group elements. In Table 2, we give analytical measurements for the computational costs of the secure DKG protocol, the partial evaluation algorithm, the combine algorithm and the verification of the pseudorandom value. The costs are computed using the number of group operations $\exp_{\mathbb{G}}, \mathsf{mul}_{\mathbb{G}}, \mathsf{pairing}$ which represent the exponentiation in $\mathbb{G}$, the multiplication in $\mathbb{G}$ and the pairing operation, respectively.

6.2 Implementation

We compare the efficiency of different DVRF implementations by benchmarking the time required to generate a single random value for DRB purposes. The protocols, and associated curves, studied in the reference implementation [Fet20] are shown in Table 3, where the protocols are implemented using mcl library [Mis19], RELIC [AG14], Libsodium [BD19b].

The results are summarised in Table 4, which shows the average time for each protocol to generate a single random value based on the average execution time of the functions PartialEval and Combine. Benchmarking was done using Catch2 [Nas19] on a laptop running Ubuntu 18.04. LTS with 64bit Intel Core i7-8550U processor, with 4GHz capacity, and 16GiB of memory. We observe that the DDH-DVRF protocol with Ristretto255 outperforms the Dfinity-DVRF protocol with curves offering the same 128-bit security level [4] by approximately a factor of 5. In the case of BN256, which has the same prime field size that Ristretto255 but lower security level, the times for random value generation are 1.5 slower if using the mcl library, and slower by over a factor of 10 if using

[4] See `https://tools.ietf.org/id/draft-yonezawa-pairing-friendly-curves-00.html` for a discussion of revised security strength of pairing-based cryptographic assumptions.

Protocol	GLOW-DVRF	DDH-DVRF	Dfinity-DVRF
PartialEval	$1 \cdot \mathbb{G}_1 + 2 \cdot \mathbb{Z}_q$	$1 \cdot \mathbb{G} + 2 \cdot \mathbb{Z}_q$	$1 \cdot \mathbb{G}_1$
Combine	$1 \cdot \mathbb{G}_1 + 1 \cdot \mathbb{Z}_q$	$1 \cdot \mathbb{G} + 2(t+1) \cdot \mathbb{Z}_q$	$1 \cdot \mathbb{G}_1 + 1 \cdot \mathbb{Z}_q$

Table 1: Size of the output of PartialEval and Combine algorithm

Operations		GLOW-DVRF	DDH-DVRF	Dfinity-DVRF
DistKG	Gen.	$(2t+3+\ell(t+2)) \cdot \mathsf{exp}_{\mathbb{G}_1}$ $+(t+1)(\ell+1) \cdot \mathsf{mul}_{\mathbb{G}_1}$	$(2t+3+\ell(t+2)) \cdot \mathsf{exp}_{\mathbb{G}}$ $+(t+1)(\ell+1) \cdot \mathsf{mul}_{\mathbb{G}}$	$(2t+3+\ell(t+2)) \cdot \mathsf{exp}_{\mathbb{G}_2}$ $+(t+1)(\ell+1) \cdot \mathsf{mul}_{\mathbb{G}_2}$
	Extra.	$(2nt+n+1) \cdot \mathsf{exp}_{\mathbb{G}_1}$ $+(3nt+2n-2t-2) \cdot \mathsf{mul}_{\mathbb{G}_1}$ $+(n-1) \cdot \mathsf{mul}_{\mathbb{G}_2} + 2n \cdot \mathsf{pairing}$	$(2nt+n+1) \cdot \mathsf{exp}_{\mathbb{G}}$ $+(3nt+2n-2t-2) \cdot \mathsf{mul}_{\mathbb{G}}$	$(2nt+n+1) \cdot \mathsf{exp}_{\mathbb{G}_2}$ $+(3nt+2n-2t-2) \cdot \mathsf{mul}_{\mathbb{G}_2}$
PartialEval		$3 \cdot \mathsf{exp}_{\mathbb{G}_1}$	$3 \cdot \mathsf{exp}_{\mathbb{G}}$	$1 \cdot \mathsf{exp}_{\mathbb{G}_1}$
Combine		$(4k+t+1) \cdot \mathsf{exp}_{\mathbb{G}_1} + t \cdot \mathsf{mul}_{\mathbb{G}_1}$	$(4k+t+1) \cdot \mathsf{exp}_{\mathbb{G}} + t \cdot \mathsf{mul}_{\mathbb{G}}$	$(t+1) \cdot \mathsf{exp}_{\mathbb{G}_1} + t \cdot \mathsf{mul}_{\mathbb{G}_1}$ $+2k \cdot \mathsf{pairing}$
Verify		$2 \cdot \mathsf{pairing}$	$5(t+1) \cdot \mathsf{exp}_{\mathbb{G}} + t \cdot \mathsf{mul}_{\mathbb{G}}$	$2 \cdot \mathsf{pairing}$

Table 2: Theoretical computational costs. Notations: ℓ represents the total number of participating servers. $n = |\mathrm{QUAL}|$ is the number of qualified servers. $k = |\mathcal{E}|$ is the number of partial evaluations with $k \geq t+1$ given to the Combine algorithm. For the DistKG sub-protocol, we compare the number of computations that each server performs in the generating phase and the extracting phase.

RELIC. Between the protocols with compact proofs, the GLOW-DVRF protocol outperforms the Dfinity-DVRF implementation on randomness generation for the same curve by at least a factor of 2.5, and the highest performing implementation, disregarding other factors, is therefore the GLOW-DVRF protocol with curve BN256. However, comparing implementations with the same security level the DDH-DVRF protocol produces the fastest times. These benchmarks are promising for distributed ledger applications, discussed in Section 6, where random value generation can impose a lower bound on block production times.

The benchmarks discussed in this section can be reproduced using the reference implementation [Fet20] that is released in Apache 2.0 license. The code provided also allows for separate benchmarking of the DistKG phase and randomness generation with message passing for all curves listed in Table 3. The communication between servers is implemented either locally, by means of a scheduler, or using network connections. For the former all nodes must reside within the same process, and can be used for running the DistKG and DRB with simulated network latency. A simple implementation of the latency, where each node's latency is sampled from a Gamma distribution with some chosen mean, is currently available. In the latter case of network connections the servers can be run on different computers identified by their IP address. Each server has an ECDSA and Diffie-Hellman key pair, which are used to sign all outgoing messages and to set up the point-to-point secure channels using Noise-C [Wea16], respectively. We assume both the ECDSA and Diffie-Hellman public keys for each server are synchronised using a third trusted party. In addition to the secure point-to-point channels, the DistKG additionally provides a secure bulletin for broadcasting information to all participants, that has been implemented using the reliable broadcast protocol described in [CKPS01], and that is of independent interest.

References

AG14. D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient Library for Cryptography. `https://github.com/relic-toolkit/relic`, 2014.

AMMR18. Shashank Agrawal, Payman Mohassel, Pratyay Mukherjee, and Peter Rindal. DiSE: Distributed Symmetric-key Encryption. In *CCS 2018*, pages 1993–2010. ACM, 2018.

Protocol	Curve	Pseudo-randomness	Security Level (bits)	Prime Field Size	Assumption
GLOW-DVRF	BN256	Strong	100	256	co-CDH XDH
	BLS12-381	Strong	128	381	
	BN384	Strong	128	384	
DDH-DVRF	Ristretto255	Strong	128	255	DDH
	Curve25519	Strong	128	255	DDH
Dfinity-DVRF	BN256	Standard	100	256	co-CDH
	BLS12-381	Standard	128	381	
	BN384	Standard	128	384	

Table 3: Curves and their associated properties examined in benchmarks

Protocol	Curve	Library	Proof size (bytes)	Randomness Generation (ms)	Time Ratio
GLOW-DVRF	BN256	mcl	79	7.38	0.69
	BLS12-381	mcl	117	18.67	1.75
	BN384	mcl	117	21.39	2.00
	BN_P256	RELIC	33	33.16	3.10
DDH-DVRF	Ristretto255	Libsodium	1664	10.70	1
	Curve25519	RELIC	1664	65.97	6.17
Dfinity-DVRF	BN256	mcl	79	18.81	1.76
	BLS12-381	mcl	117	55.79	5.22
	BN384	mcl	117	60.73	5.68
	BN_P256	RELIC	33	138.36	12.94

Protocol	Curve	Library	Proof size (bytes)	Randomness Generation (ms)	Time Ratio
GLOW-DVRF	BN256	mcl	79	29.06	0.68
	BLS12-381	mcl	117	71.91	1.68
	BN384	mcl	117	80.15	1.87
	BN_P256	RELIC	33	132.92	3.10
DDH-DVRF	Ristretto255	Libsodium	6464	42.91	1
	Curve25519	RELIC	6464	285.59	6.66
Dfinity-DVRF	BN256	mcl	79	74.01	1.72
	BLS12-381	mcl	117	215.34	5.02
	BN384	mcl	117	228.54	5.33
	BN_P256	RELIC	33	566.21	13.20

Table 4: Time ratios per individual node for each protocol in Table 3 to generate one round of entropy for total number of servers $\ell = 50$ (upper table) and $\ell = 200$ (lower table), with threshold value $t = \ell/2$. The time ratio is computed as the time to generate a single random value for each protocol divided by the observed time for DDH-DVRF, which is the DVRF offering higher security, implemented using LibSodium.

BD19a. Razvan Barbulescu and Sylvain Duquesne. Updating key size estimations for pairings. *J. Cryptology*, 32(4):1298–1336, 2019.

BD19b. Daniel J. Bernstein and Frank Denis. Libsodium - a modern, portable, easy to use crypto library. `https://github.com/jedisct1/libsodium`, 2019.

BDN18. Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *ASIACRYPT 2018*, pages 435–464, 2018.

BKM18. Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *CoRR*, abs/1807.04938, 2018.

BLMR13. Dan Boneh, Kevin Lewi, Hart Montgomery, and Ananth Raghunathan. Key Homomorphic PRFs and Their Applications. In *CRYPTO 2013*, pages 410–428, 2013.

BLS01. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *Advances in Cryptology - ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, 2001.

Bol02. Alexandra Boldyreva. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *Public Key Cryptography — PKC 2003*, pages 31–46, 2002.

But18. Vitalik Buterin. Ethereum 2.0 Mauve Paper. `https://cdn.hackaday.io/files/10879465447136/Mauve%20Paper%20Vitalik.pdf`, 2018.

CD17. Ignacio Cascudo and Bernardo David. SCRAPE: scalable randomness attested by public entities. In *Applied Cryptography and Network Security - ACNS*, volume 10355 of *Lecture Notes in Computer Science*, pages 537–556. Springer, 2017.

CHL05. Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In *EUROCRYPT 2005*, pages 302–321, 2005.

CKPS01. Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Advances in Cryptology — CRYPTO 2001*, pages 524–541, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

Clo19. Cloudflare. Decentralized Verifiable Randomness Beacon. `https://developers.cloudflare.com/randomness-beacon/`, 2019.

Cor19. Corestar. Corestar Arcade: Tendermint-based Byzantine Fault Tolerant (BFT) middleware with an embedded BLS-based random beacon. `https://github.com/corestario/tendermint`, 2019.

CP93. David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *CRYPTO' 92*, pages 89–105, 1993.

DAO19. DAOBet. DAOBet (ex — DAO.Casino) to Deliver On-Chain Random Beacon Based on BLS Cryptography. `https://daobet.org/blog/on-chain-random-generator/`, 2019.

DGKR18. Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Advances in Cryptology - EUROCRYPT 2018, Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 66–98. Springer, 2018.

Dod03. Yevgeniy Dodis. Efficient construction of (distributed) verifiable random functions. In *Public Key Cryptography - PKC 2003*, volume 2567, pages 1–17. Springer, 2003.

DSD07. Augusto Jun Devegili, Michael Scott, and Ricardo Dahab. Implementing Cryptographic Pairings over Barreto-Naehrig Curves. In *Pairing*, pages 197–207, 2007.

DY05. Yevgeniy Dodis and Aleksandr Yampolskiy. A Verifiable Random Function with Short Proofs and Keys. In *Public Key Cryptography - PKC 2005*, pages 416–431, 2005.

Fet20. Fetch.ai. Distributed Verifiable Random Functions: an Enabler of Decentralized Random Beacons. `https://github.com/fetchai/research-dvrf`, 2020.

FZ12. Matthew K. Franklin and Haibin Zhang. Unique Group Signatures. In *ESORICS 2012*, volume 7459, pages 643–660, 2012.

GHM+17. Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 51–68. ACM, 2017.

GJKR07. 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.

GJNB11. C. C. F. Pereira Geovandro, Marcos A. Simplício 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.

GO93. Shafi Goldwasser and Rafail Ostrovsky. Invariant signatures and non-interactive zero-knowledge proofs are equivalent (extended abstract). In *Advances in Cryptology - CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 228–245. Springer, 1993.

GRPV18. Sharon Goldberg, Leonid Reyzin, Dimitrios Papadopoulos, and Jan Včelák. Verifiable Random Functions (VRFs). Internet-Draft draft-irtf-cfrg-vrf-03, Internet Engineering Task Force, September 2018. Work in Progress.

HMW18. Timo Hanke, Mahnush Movahedi, and Dominic Williams. DFINITY technology overview series, consensus system. *CoRR*, abs/1805.04548, 2018.

Kee19. Keep. The Keep Random Beacon: An Implementation of a Threshold Relay. `http://docs.keep.network/random-beacon/`, 2019.

KJG+18. Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In *IEEE Symposium on Security and Privacy*, pages 583–598, 2018.

KM13. Veronika Kuchta and Mark Manulis. Unique aggregate signatures with applications to distributed verifiable random functions. In *CANS 2013*, volume 8257, pages 251–270, 2013.

MBB+15. Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. CONIKS: Bringing Key Transparency to End Users. In *24th USENIX Security Symposium*, pages 383–398, 2015.

Mis19. Shigeo Mistunari. mcl - A portable and fast pairing-based cryptography library. `https://github.com/herumi/mcl`, 2019.

MRV99. Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *FOCS '99*, pages 120–130. IEEE Computer Society, 1999.

Nak08. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. `http://bitcoin.org/bitcoin.pdf`.

Nas19. Phil Nash. Catch2. `https://github.com/catchorg/Catch2`, 2019.

Ped91. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO 1991*, pages 129–140, 1991.

SJSW19. Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar R. Weippl. ETHDKG: distributed key generation with ethereum smart contracts. *IACR Cryptology ePrint Archive*, 2019:985, 2019.

SJSW20. Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar R. Weippl. Hydrand: Practical continuous distributed randomness. *IEEE S&P*, 2020.

Wea16. Rhys Weatherley. Noise-C, a plain C implementation of the Noise protocol. `https://github.com/rweather/noise-c`, 2016.

Woo19. Gavin Wood. Ethereum: Secure decentralised generalised and transaction ledger, 2019. `https://ethereum.github.io/yellowpaper/paper.pdf`.

A Secure DKG protocols

We recall in Figure 1 the DKG protocol originally proposed in [GJKR07]. This protocol is used in our proposed DDH-based DVRF called DDH-DVRF. The DKG protocol used in Dfinity-DVRF can be found in Figure 2, whereas the protocol used in GLOW-DVRF is given in Figure 3.

B Security proofs for DDH-DVRF

Theorem 1. $\mathcal{V}^{\mathsf{DDH\text{-}DVRF}}$ is admissible and correct.

– **Generating phase:**

1. Each server S_i performs a Pedersen-VSS of a random value $a_{i,0}$:
   (a) S_i chooses two random t-degree polynomials f_i, f'_i over $\mathbb{Z}_q$:

$$f_i(z) = a_{i,0} + a_{i,1}z + \cdots + a_{i,t}z^t \qquad f'_i(z) = b_{i,0} + b_{i,1}z + \cdots + b_{i,t}z^t$$

   S_i broadcasts $C_{i,k} = g^{a_{i,k}}h^{b_{i,k}}$ for $0 \le k \le t$. S_i computes the shares $s_{i,j} = f_i(j), s'_{i,j} = f'_i(j)$ for $1 \le j \le \ell$ and sends $s_{i,j}, s'_{i,j}$ to server S_j.
   (b) Each server S_j verifies the shares he received from the other servers. S_j checks if

$$g^{s_{i,j}}h^{s'_{i,j}} = \prod_{k=0}^{t}(C_{i,k})^{j^k} \tag{1}$$

   for $1 \le i \le \ell$. If the check fails for an index i, S_j broadcasts a complaint against S_i.
   (c) Each server S_i who received a complaint from server S_j broadcasts the values $s_{i,j}, s'_{i,j}$ that satisfy Equation 5.
   (d) Each server marks as *disqualified* any server that either
   - Received more than t complaints in Step 1b, or
   - Answered to a complaint in Step 1c with values that falsify Equation 5.
2. Each server then builds the set of non-disqualified players QUAL.
3. Each server S_i sets his share of the secret as $\mathsf{sk}_i = \sum_{j\in\mathrm{QUAL}} s_{j,i}$, the verification key $\mathsf{vk}_i = g^{\mathsf{sk}_i}$ and the value $r_i = \sum_{j\in\mathrm{QUAL}} s'_{j,i}$.

– **Extracting phase:**

4. Each server $i \in$ QUAL exposes $A_{i,0} = g^{a_{i,0}}$ via **Feldman-VSS**
   (a) Each server S_i with $i \in$ QUAL, broadcasts $A_{i,k} = g^{a_{i,k}}$ for $0 \le k \le t$.
   (b) Each server S_j verifies the values broadcast by the other servers in QUAL, formally, for each $i \in$ QUAL, S_j checks if

$$g^{s_{i,j}} = \prod_{k=0}^{t}(A_{i,k})^{j^k} \tag{2}$$

   If the check fails for some index i, S_j complains against S_i by broadcasting the values $s_{i,j}, s'_{i,j}$ that satisfy Equation 5 but not Equation 6.
   (c) For server S_i who receives at least one valid complaint, i.e., values which satisfy Equation 5 but not Equation 6, the other servers run the re-construction phase of **Pedersen-VSS** to compute $a_{i,0}, f_i, A_{i,k}$ for $0 \le k \le t$. Compute the global public key as $\mathsf{pk} = \prod_{i\in\mathrm{QUAL}} A_{i,0}$. The distributed secret value $\mathsf{sk} = \sum_{i\in\mathrm{QUAL}} a_{i,0}$ is not explicitly computed by any party.
   (d) Each server S_i can reconstruct the verification keys of other servers:
   - Compute $v_k = \prod_{i\in\mathrm{QUAL}} A_{i,k}$ for $0 \le k \le t$.
   - For each $j \in$ QUAL such that $j \ne i$, compute $\mathsf{vk}_j = \prod_{k=0}^{t} v_k^{j^k}$.

Fig. 1: Secure distributed key generation (DKG) protocol [GJKR07] used in DDH-DVRF.

Proof. First, we show that $\mathcal{V}^{\mathsf{DDH\text{-}DVRF}}$ is admissible. To see that consistency and domain range correctness are satisfied by $\mathcal{V}^{\mathsf{DDH\text{-}DVRF}}$ it suffices to see that the following equality holds:

$$\sum_{j\in\Delta} \mathsf{sk}_j \lambda_{0,j,\Delta} = \sum_{j\in\Delta} \lambda_{0,j,\Delta} \left(\sum_{i\in \mathrm{QUAL}} s_{i,j} \right) = \sum_{i\in\mathrm{QUAL}} \left(\sum_{j\in\mathcal{I}} \lambda_{0,j,\Delta} \cdot s_{i,j} \right)$$
$$= \sum_{i\in\mathrm{QUAL}} \left(\sum_{j\in\Delta} \lambda_{0,j,\Delta} \cdot f_i(j) \right) = \sum_{i\in\mathrm{QUAL}} a_{i,0} = \mathsf{sk}$$

Then $\prod_{j\in\Delta}(H_1(x)^{\mathsf{sk}_j})^{\lambda_{0,j,\Delta}} = H_1(x)^{\left(\sum_{j\in\Delta}\lambda_{0,j,\Delta}\cdot\mathsf{sk}_j\right)} = H_1(x)^{\mathsf{sk}}$ holds for every subset $\Delta \subseteq \{1,\ldots,\ell\}$ with $|\Delta| = t+1$.

Domain range correctness is obvious and is omitted. Provability follows from the completeness property of the NIZKs and the consistency of the DVRF. Uniqueness can be proven using the extractability of the NIZKs. That is, for any (v,π), if π verifies, we can extract k such that $v = H_1(x)^k$ and $\mathsf{vk} = g^k$.

Next, we shall prove the $\mathcal{V}^{\mathsf{DDH\text{-}DVRF}}$ scheme satisfies the correctness defined in 9. The proof relies on the extractibility property of NIZKs. We first describe the original correctness game in the random oracle model:

$\underline{\mathsf{Correct}_{\mathcal{A}}(b)}$:

1. Give the public parameters $(\mathbb{G}, q, g)$ and a list of servers $S = \{1, \cdots, \ell\}$ to the adversary $\mathcal{A}$
2. $\mathcal{A}$ chooses to corrupt a collection C of servers with $|C| \leq t$. Without loss of generality, let $C = \{1, \cdots, m\}$. Get the set C from $\mathcal{A}$.
3. Run the distributed key generation protocol $\mathsf{DistKG}(1^k, t, \ell)$ with ℓ servers S. The protocol outputs a set QUAL of qualified servers, a key pair $(\mathsf{sk}_i, \mathsf{vk}_i)$ for every honest server $S_i \in \mathrm{QUAL} \setminus C$, and outputs key pair $(\mathsf{sk}_i, \mathsf{vk}_i)$ in which one of keys may be undefined for corrupted servers $S_i \in C$.
4. The random oracle H_1 is programmed as follows: Define a list $\mathcal{L}_{H_1} = \emptyset$. For a query on x:
   (a) If there exists $(x, r, h) \in \mathcal{L}_{H_1}$, then the oracle outputs h.
   (b) Otherwise, choose a random $r \xleftarrow{R} \mathbb{Z}_q$ and set $h = g^r$ and update the list $\mathcal{L}_{H_1} = \mathcal{L}_{H_1} \cup (x, r, h)$. The oracle outputs h.
5. The random oracle H_2 is programmed as follows: Define a list $\mathcal{L}_{H_2} = \emptyset$. For a query on y, choose a random $c \xleftarrow{R} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup (y, c)$. The oracle outputs c.
6. On an evaluation query (Eval, x, i) for an honest server $S_i \in \mathrm{QUAL} \setminus C$, compute $H_1(x)^{\mathsf{sk}_i}$ and run PrEq_{H_2} to generate a proof π_i, and return $(i, H_1(x)^{\mathsf{sk}_i}, \pi_i)$. Otherwise return $\perp$.
7. On the challenge query $(\mathsf{Challenge}, x^\star, U, \{(i, v_i^\star, \pi_i^\star)\}_{i\in U\cap C})$ with $U \subseteq \mathrm{QUAL}$:
   (a) If $|U| \leq t$ or any of $\pi_i^\star$ do not verify, output 0
   (b) Else, compute $v_j = H_1(x^\star)^{\mathsf{sk}_j}$ for $j \in U$ and $v_j^\star = H_1(x^\star)^{\mathsf{sk}_j}$ for $j \in U \setminus C$. Let $v = \prod_{j\in U} v_j^{\lambda_{0,j,U}}$ and $v^\star = \prod_{j\in U} v_j^{\star\lambda_{0,j,U}}$. If $v^\star = v$, output 0.
   (c) Else, output 1.

Suppose there exists an adversary $\mathcal{A}$ leading the challenger to output 1 with non-negligible probability. We will show that this leads to a contradiction. When the challenger outputs 1, all the proofs $\{\pi_i^\star\}_{i\in U\cap C}$ received from the adversary are verified but $v^\star \neq v$. This means there exists $i \in U$ such that $v_i^\star \neq v_i$. If $i \in U \setminus C$, it is impossible since $v_i^\star = v_i = H_1(x^\star)^{\mathsf{sk}_i}$. If $i \in U \cap C$, use the extractor of the NIZKs to obtain a witness sk_i' such that $\mathsf{vk}_i = g^{\mathsf{sk}_i'}$ and $v_i^\star = H_1(x^\star)^{\mathsf{sk}_i'}$. Since $v_i^\star \neq v_i$, we know that $\mathsf{sk}_i' \neq \mathsf{sk}_i$ which leads to $\mathsf{vk}_i \neq g^{\mathsf{sk}_i}$ which contradicts to the premise.

Next we give a proof for the strong pseudorandomness of DDH-based DVRF. Our proof is constructed using the simulator for proving secrecy of the secure DKG protocol in [GJKR07] and the proofs of Theorem 8.1 and 8.2 in [AMMR18]. In addition, we use standard zero-knowledge property to simulate NIZKs.

Theorem 2. $\mathcal{V}^{\mathsf{DDH\text{-}DVRF}}$ is strongly pseudorandom under the *DDH* assumption in the random oracle model.

Proof. For any PPT adversary, we shall first describe the real game $\mathsf{PseudoRand}_{\mathcal{A}}(b)$ as below.

$\underline{\mathsf{PseudoRand}_{\mathcal{A}}(b)}$:

1. Give the public parameters $(\mathbb{G}, q, g)$ and a list of servers $S = \{1, \cdots, \ell\}$ to the adversary $\mathcal{A}$
2. $\mathcal{A}$ chooses to corrupt a collection C of servers with $|C| \leq t$. Without loss of generality, let $C = \{1, \cdots, m\}$. Get the set C from $\mathcal{A}$.
3. Run the distributed key generation protocol $\mathsf{DistKG}(1^k, t, \ell)$ with ℓ servers S. The protocol outputs a set QUAL of qualified servers with QUAL $\subseteq S$, key pair $(\mathsf{sk}_i, \mathsf{vk}_i)$ for every honest server $i \in \text{QUAL} \backslash C$, and outputs key pair $(\mathsf{sk}_i, \mathsf{vk}_i)$ in which one of keys may be undefined for corrupted servers $i \in C$.
4. The random oracle H_1 is programmed as follows: Define a list $\mathcal{L}_{H_1} = \emptyset$. For a query on x:
   (a) If there exists $(x, r, h) \in \mathcal{L}_{H_1}$, then the oracle outputs h.
   (b) Otherwise, choose a random $r \xleftarrow{R} \mathbb{Z}_q$ and set $h = g^r$ and update the list $\mathcal{L}_{H_1} = \mathcal{L}_{H_1} \cup (x, r, h)$. The oracle outputs h.
5. The random oracle H_2 is programmed as follows: Define a list $\mathcal{L}_{H_2} = \emptyset$. For a query on y, choose a random $c \xleftarrow{R} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup (y, c)$. The oracle outputs c.
6. On an evaluation query (Eval, x, i) for an honest server $i \in \text{QUAL} \setminus C$, compute $H_1(x)^{\mathsf{sk}_i}$ and run PrEq_{H_2} to generate a proof π_i, and return $(H_1(x)^{\mathsf{sk}_i}, \pi_i)$. Otherwise return $\perp$.
7. On the challenge query $(\mathsf{Challenge}, x^\star, \{(i, v_i^\star, \pi_i)\}_{i \in U}, V)$ with $|V| > t$ and $V \subseteq$ QUAL and $U \subseteq V \cap C$,
   (a) If $\mathcal{A}$ has made at least $t + 1 - |C|$ queries of the form $(\mathsf{Eval}, x^\star, *)$, then output 0 and stop.
   (b) Otherwise do as follows:
      i. Run $\mathsf{VerifyEq}_{H_1}$ to check the proofs π_i for $i \in U$. If any check fails, output 0 and stop.
      ii. Set $v_i^\star = H_1(x^\star)^{\mathsf{sk}_i}$ for $i \in V \setminus U$.
      iii. Compute $v^\star = \prod_{i \in V} {v_i^\star}^{\lambda_{0,i,V}}$.
      iv. If $v^\star = \perp$ then return $\perp$. Otherwise depending on b do as follows:
         A. If $b = 0$ then return $v^\star$;
         B. Else return a uniform random.
8. Continue answering evaluation queries as before, but if $\mathcal{A}$ makes queries of the form $(\mathsf{Eval}, x^\star, i)$ for some $i \in \text{QUAL} \setminus C$ and i is the $(t + 1 - |C|)$-th honest server that $\mathcal{A}$ contacted then output 0 and stop.
9. Receive a guess b' from $\mathcal{A}$ and output b'.

First hybrid - $\mathsf{Hyb}^{zk}_{\mathcal{A}}(b)$ Define a hybrid $\mathsf{Hyb}^{zk}_{\mathcal{A}}(b)$ which is similar to $\mathsf{PseudoRand}_{\mathcal{A}}(b)$ except that real proofs π_i in Step 6 and 8 are replaced with simulated proofs. $\mathsf{Hyb}^{zk}_{\mathcal{A}}(b)$ is indistinguishable from $\mathsf{PseudoRand}_{\mathcal{A}}(b)$ for any PPT $\mathcal{A}$ and $b \in \{0, 1\}$ due to the zero-knowledge property of NIZKs.

Second hybrid - $\mathsf{Hyb}^{sim}_{\mathcal{A}}(b)$ Define a hybrid $\mathsf{Hyb}^{sim}_{\mathcal{A}}(b)$ which is similar to $\mathsf{Hyb}^{zk}_{\mathcal{A}}(b)$ except that the running of the secure DKG protocol $\mathsf{DistKG}(1^k, t, \ell)$ in Step 3 is replaced with a simulator described in [GJKR07] which sets the public key pk to a given element $Y = g^\alpha$ with α unknown. We give a brief description of the construction of the simulator below:

1. Generating phase: The simulator runs on behalf of all the honest servers. It performs the generating phase exactly as in the secure DKG protocol. The simulator chooses random polynomials $f_i(z) = a_{i,0} + a_{i,1}z + \cdots + a_{i,t}z^t$ and $f'_i(z) = b_{i,0} + b_{i,1}z + \cdots + b_{i,t}z^t$ for each honest server $i \in S \setminus C$. At the end of the generating phase of the secure DKG protocol, the simulator obtains a well-defined set of non-disqualified servers QUAL which contains all the honest servers and some corrupted servers. The simulator receives all the shares $s_{i,j}, s'_{i,j}$ from the corrupted servers and thus can recover all the polynomials $f_i(z), f'_i(z)$ for $i \in \text{QUAL} \cap C$.
2. Extracting phase:
   - For the honest servers $i \in \text{QUAL} \setminus (C \cup \{\ell\})$, compute $A_{i,j} = g^{a_{i,j}}$ for $0 \le j \le t$.
   - For the ℓ-th honest server, the simulator replaces f_ℓ with a new polynomial $f^\star_\ell$ which is determined by the values: $\alpha - \sum_{j \in \text{QUAL}\setminus\{\ell\}} f_j(0)$, $f_\ell(1)$, $\ldots$, $f_\ell(t)$. The simulator can compute $A^\star_{\ell,j}$ for $0 \le j \le t$ based on the new polynomial $f^\star_\ell(x)$ without knowing the value of α:
     - $A^\star_{\ell,0} = Y \cdot \prod_{i \in \text{QUAL}\setminus\{\ell\}} A^{-1}_{i,0}$
     - $A^\star_{\ell,j} = (A^\star_{\ell,0})^{\delta_{0,j}} \prod_{i=1}^{t} g^{f_\ell(i)\cdot\delta_{i,j}}$ where $\delta_{i,j}$ is the coefficient of x^j in the lagrange basis polynomial $\lambda_{i,T}(x)$ with $T = \{0, 1, \ldots, t\}$.

   Broadcast $A_{i,j}$, $A^\star_{\ell,j}$ for $i \in \text{QUAL} \setminus \{\ell\}$ and $0 \le j \le t$.

Therefore, the final polynomial $f^\star$ takes the values $f^\star(0) = \alpha$, $f^\star(1) = f(1)$, $\ldots$, $f^\star(t) = f(t)$. The global public key $\mathsf{pk} = A^\star_{\ell,0} \prod_{i \in \text{QUAL}\setminus\{\ell\}} A_{i,0} = g^\alpha$.

For each honest server $j \in \text{QUAL} \setminus C$ and $j \le t$, the simulator can compute j's secret key $\mathsf{sk}_j = f^\star(j)$ as well as the corresponding verification key vk_j. For each honest server $j \in \text{QUAL} \setminus C$ and $j > t$, the simulator is not able to computer j's secret key because $f^\star(0) = \alpha$ is unknown. This means the simulator cannot compute $f^\star(j)$ for $j > t$. However, the simulator can still compute the corresponding verification keys $\mathsf{vk}_j = \mathsf{pk}^{\lambda_{j,0,T}} \cdot \prod_{i=1}^{t} \mathsf{vk}_i^{\lambda_{j,i,T}}$ for $j > t$ where $T = \{0, 1, \ldots, t\}$ and $\lambda_{j,i,T}$ are the lagrange coefficients.

Although the simulator does not have sk_i for $i \in \text{QUAL} \setminus C$ and $i > t$, with the value of $\mathsf{vk}_i = g^{\mathsf{sk}_i}$ it is sufficient to answer all the evaluation queries in Step 6, 7 and 8 defined in the game $\mathsf{PseudoRand}_{\mathcal{A}}(b)$. In the game $\mathsf{Hyb}^{sim}_{\mathcal{A}}(b)$, we modify all the computation of $v_i = H_1(x)^{\mathsf{sk}_i}$ for $i \in \text{QUAL} \setminus C$ and $i > t$ as follows: assume $(x, r, h) \in L_{H_1}$, compute $v_i = \mathsf{vk}_i^r$. The corresponding NIZK that shows v_i is correctly formed can be simulated using random oracle H_2 without knowing sk_i. Finally $\mathsf{Hyb}^{sim}_{\mathcal{A}}(b)$ is indistinguishable from $\mathsf{Hyb}^{zk}_{\mathcal{A}}(b)$ for any PPT $\mathcal{A}$ and $b \in \{0, 1\}$ because the probability distribution of the output of the simulator is identical to the original secure DKG protocol [GJKR07].

k-hybrids - $\mathsf{Hyb}^k_{\mathcal{A}}(b)$ For any adversary $\mathcal{A}$ that asks evaluation/challenge queries on q_E distinct x, we define hybrid games $\mathsf{Hyb}^k_{\mathcal{A}}(b)$. The hybrid games $\mathsf{Hyb}^k_{\mathcal{A}}(b)$ are exactly the same as $\mathsf{Hyb}^{sim}_{\mathcal{A}}(b)$ except how the evaluation/challenge queries are answered on the first k distinct plaintext x. The simulator runs the secure DKG protocol to set up the system and assume the final polynomial is $f^\star$. The simulator in addition chooses k t-degree random polynomials $f^1, f^2, \cdots, f^k$ such as for all corrupted servers $i \in C$, $f^j(i) = f^\star(i)$, for $1 \le j \le k$. That is, the random polynomials match on the shares of the corrupted servers.

Now an evaluation query (Eval, x, i) for an honest i is answered as follows: if x is the j-th distinct plaintext in the evaluation/challenge queries, then return $\mathsf{pc}(x, j, i)$ defined as follows:

$$\mathsf{pc}(x, j, i) = \begin{cases} (H_1(x)^{f^j(i)}, \pi_i) \text{ if } j \le k \\ (H_1(x)^{\mathsf{sk}_i}, \pi_i) \quad \text{otherwise} \end{cases}$$

where π_i is a simulated proof generated by calling the random oracle H_2 and $H_1(x)^{\mathsf{sk}_i}$ is obtained in the same way as described in $\mathsf{Hyb}^{sim}_{\mathcal{A}}(b)$.

The challenge query $(\mathsf{Challenge}, x^\star, \{(i, z^\star_i, \pi_i)\}_{i \in U}, V)$ with $|V| > t$ and $V \subseteq \text{QUAL}$ and $U \subseteq V \cap C$ and $z^\star_i$ a set of evaluation shares from the corrupted servers, is answered as follows:

1. if $x^\star$ was queried in the evaluation phase and it was the j-th distinct value, then let $j^\star = j$. Else, assume there are so far j' distinct evaluation queries and let $j^\star = j' + 1$.
2. If $\mathcal{A}$ has made at least $t + 1 - |C|$ queries of the form $(\mathsf{Eval}, x^\star, \cdot)$, then output 0 and stop.
3. Otherwise do as follows:
   (a) Run $\mathsf{VerifyEq}_{H_2}$ to check the proofs π_i for $i \in U$. If any check fails, output 0 and stop.
   (b) Set $(z_i^\star, \cdot) = \mathsf{pc}(x^\star, j^\star, i)$ for $i \in V \setminus C$.
   (c) Compute $v^\star = \prod_{i \in V} z_i^{\star \lambda_{0,i,V}}$.
   (d) If $v^\star = \perp$ then return $\perp$. Otherwise depending on b do as follows:
      i. If $b = 0$ then return $v^\star$;
      ii. Else return a uniform random.

Obviously, $\mathsf{Hyb}_{\mathcal{A}}^0(b)$ is identical to $\mathsf{Hyb}_{\mathcal{A}}^{sim}(b)$. $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^k(b)$ only differs in the evaluation queries for the k-th distinct x. Below we shall prove that $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^k(b)$ are indistinguishable from the attacker's point of view.

Lemma 1. *For any $b \in \{0,1\}$ and $1 \le k \le q_E$, the outputs of hybrids $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^k(b)$ are computationally indistinguishable under DDH assumption.*

Proof. We show that if there exists a PPT adversary $\mathcal{A}$ that can distinguish between the hybrids $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^k(b)$ with non-negligible probability then we can construct a PPT adversary $\mathcal{B}$ to break an extended version of the DDH assumption using $\mathcal{A}$ as a subroutine.

The extended DDH problem [AMMR18] is given by

$$(\mathbb{G}, q, g, g^{\alpha_0}, g^{\alpha_1}, \cdots, g^{\alpha_w}, g^\beta, y_0, y_1, \cdots, y_w)$$

where $y_i = g^{\alpha_i \beta}$ for all $i \in \{0, 1, \ldots, w\}$ or randoms. The extended DDH problem can be easily derived from the original DDH problem. We now construct $\mathcal{B}$ using $\mathcal{A}$ as follows:

1. Give the public parameters $(\mathbb{G}, q, g)$ and a list of servers $S = \{1, 2, \ldots, \ell\}$ to $\mathcal{A}$
2. $\mathcal{A}$ chooses a set C of servers with $|C| \le t$ to corrupt and send C to $\mathcal{B}$. W.l.o.g., assume $C = \{1, 2, \cdots, m\}$.
3. The random oracle H_1 is answered as follows: initialise $\mathcal{L}_{H_1} = \emptyset$. Let q_{H_1} be the total number of distinct random oracle queries asked in this game. Choose an index $\eta^\star \xleftarrow{R} [q_{H_1}]$ uniformly at random.
   – If there exists a tuple $(x, r, h) \in \mathcal{L}_{H_1}$, output h.
   – Otherwise,
      • if this is the $\eta^\star$-th distinct call, set $r = \perp$ and $h = g^\beta$ where g^β is from the DDH problem.
      • else choose a random $r \xleftarrow{R} \mathbb{Z}_q$ and set $h = g^r$.

   Update $\mathcal{L}_{H_1} = \mathcal{L}_{H_1} \cup (x, r, h)$

   Give the random oracle access to $\mathcal{A}$.
4. The random oracle H_2 is programmed as follows: Define a list $\mathcal{L}_{H_2} = \emptyset$. For a query on y, choose a random $c \xleftarrow{R} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup (y, c)$. The oracle outputs c. Give the random oracle access to $\mathcal{A}$.
5. Run the simulator of the secure DKG protocol to interact with the corrupted servers. $\mathcal{B}$ acts on behalf of all the honest servers $\{m+1, \ldots, \ell\}$, while the adversary controls all the corrupted servers $\{1, \ldots, m\}$. The generating phase is run exactly the same as described in Figure 1. The generating phase determines a set of non-disqualified servers QUAL which contains all the honest servers and some of the corrupted servers. Since $\mathcal{B}$ runs all the honest servers, $\mathcal{B}$ obtains all the shares from the corrupted servers and can recover all the polynomials f_i with $i \in \text{QUAL} \cap C$ chosen by the adversary. Assume the combined polynomial after the generating phase is f. In the extracting phase, the last polynomial, i.e., the ℓ-th polynomial $f_\ell^\star$, is constructed to take

the values: $\alpha_0 - \sum_{j\in \text{QUAL}\setminus\{\ell\}} f_j(0),\ f_\ell(1),\ \ldots,\ f_\ell(m),\ \alpha_{m+1} - \sum_{j\in \text{QUAL}\setminus\{\ell\}} f_j(m+1),\ \ldots,\ \alpha_t - \sum_{j\in \text{QUAL}\setminus\{\ell\}} f_j(t)$. For $0 \le j \le t$, we show how to compute $A_{i,j}$ for $i \in \text{QUAL} \setminus \{\ell\}$ and $A^\star_{\ell,j}$:

- For $i \in \text{QUAL} \setminus \{\ell\}$, the simulator has all the coefficients $a_{i,j}$ for $0 \le j \le t$, thus $A_{i,j} = g^{a_{i,j}}$ can be easily computed.
- For the ℓ-th polynomial $f^\star_\ell$, the simulator sets $A^\star_{\ell,0} = g^{\alpha_0} \prod_{i\in \text{QUAL}\setminus\{\ell\}} A_{i,0}^{-1}$ and $A^\star_{\ell,j} = (A^\star_{\ell,0})^{\delta_{0,j}} \prod_{i=1}^{m} g^{f_\ell(i)\cdot\delta_{i,j}} \prod_{i=m+1}^{t} (g^{\alpha_i} \prod_{k\in \text{QUAL}\setminus\{\ell\}} g^{-f_k(i)})^{\delta_{i,j}}$ where $\delta_{i,j}$ is the coefficient of x^j in the lagrange basis polynomial $\lambda_{i,T}(x)$ with $T = \{0, 1, \ldots, t\}$.

Let the final combined polynomial be $f^\star$. We can see that $f^\star(0) = \alpha_0$, $f^\star(1) = f(1), \ldots, f^\star(m) = f(m)$, $f^\star(m+1) = \alpha_{m+1}, \ldots, f^\star(t) = \alpha_t$. This means the shares of the honest servers $m+1 \le i \le t$ are set to be $\mathsf{sk}_i = \alpha_i$ and the corresponding verification keys are $\mathsf{vk}_i = g^{\alpha_i}$ despite $\alpha_0, \alpha_{m+1}, \ldots, \alpha_t$ are unknown. For the honest servers $t+1 \le i \le \ell$, we also cannot compute their shares sk_i because $\alpha_0, \alpha_{m+1}, \ldots, \alpha_t$ are unknown, but we can compute their verification keys as $\mathsf{vk}_i = g^{\alpha_0 \lambda_{0,0,T}} \prod_{j\in T, j\neq 0} g^{f^\star(j)\lambda_{i,j,T}}$ with $T = \{0, \ldots, t\}$. It is not difficult to verify that the outputs of the simulator have the same probability distribution as the outputs of the secure DKG protocol using a similar argument as in [GJKR07].

6. Choose $k-1$ t-degree random polynomials, $f^1, f^2, \ldots, f^{k-1}$ such that for all $i \in C$ and $1 \le j \le k-1$, $f^j(i) = f^\star(i)$.
7. Compute $\bar{z}_i = (g^\beta)^{f(i)}$ for $i \in C$. Set $\bar{z}_0 = y_0$ and $\bar{z}_i = y_i$ for $m+1 \le i \le t$ where $y_0, y_{m+1}, \ldots, y_t$ are from the DDH problem. Then for all $m+1 \le i \le \ell$, compute $\bar{z}_i = \prod_{j\in T} \bar{z}_j^{\lambda_{i,j,T}}$ with $T = \{0, 1, \ldots, t\}$.
8. Define a function $\mathsf{pc}(x, j, i)$ as follows: invoke random oracle H_1 to get $(x, r, h) \in L_{H_1}$ and return
   - $(h^{f^j(i)}, \pi_i)$ if $j < k$
   - $(\bar{z}_i, \pi_i)$ if $j = k$
   - (vk_i^r, π_i) if $j > k$ (if $r = \perp$, return $\perp$)

   where π_i is a simulated proof generated by calling the random oracle H_2.
9. On an evaluation query (Eval, x, i) for an honest $i \in \text{QUAL} \setminus C$, if x is the j-th distinct value, then return $\mathsf{pc}(x, j, i)$.
10. On the challenge query $(\mathsf{Challenge}, x^\star, \{(i, z_i^\star, \pi_i)\}_{i\in U}, V)$, where $|V| > t$ and $V \subseteq \text{QUAL}$ and $U \subseteq V \cap C$ and $z_i^\star$ a set of evaluation shares from the corrupted servers, is answered as follows:
    (a) if $x^\star$ was queried in the evaluation phase and it was the j-th distinct value, then let $j^\star = j$. Else, assume there are so far j' distinct evaluation queries and let $j^\star = j' + 1$.
    (b) If $\mathcal{A}$ has made at least $t + 1 - m$ queries of the form $(\mathsf{Eval}, x^\star, \cdot)$, then output 0 and stop.
    (c) Otherwise do as follows:
        i. Run $\mathsf{VerifyEq}_{H_2}$ to check the proofs π_i for $i \in U$. If any check fails, output 0 and stop.
        ii. Set $(z_i^\star, \cdot) = \mathsf{pc}(x^\star, j^\star, i)$ for $i \in V \setminus C$.
        iii. Compute $v^\star = \prod_{i\in V} z_i^{\star\lambda_{0,i,V}}$.
        iv. If $v^\star = \perp$ then return $\perp$. Otherwise depending on b do as follows:
            A. If $b = 0$ then return $v^\star$;
            B. Else return a uniform random.
11. Continue answering evaluation queries as before, but if $\mathcal{A}$ makes queries of the form $(\mathsf{Eval}, x^\star, i)$ for some $i \in \text{QUAL} \setminus C$ and i is the $(t+1-m)$-th honest server that $\mathcal{A}$ contacted then output 0 and stop.
12. Receive a guess b' from $\mathcal{A}$ and output b'.

Suppose the k-th distinct evaluation query x_k is the η-th query for random oracle H_1. Let's consider the case when $\eta^\star = \eta$. pc in Step 8 never returns $\perp$ because r is set to $\perp$ only for the $\eta^\star$-th call and we have assumed that this call is for x_k.

- If $y_0 = g^{\alpha_0\beta}, y_{m+1} = g^{\alpha_{m+1}\beta}, \ldots, y_\ell = g^{\alpha_\ell\beta}$, pc returns $\bar{z}_i$ on x_k which is equal to $H_1(x_k)^{\mathsf{sk}_i}$. In this case, $\mathcal{B}$ simulates $\mathsf{Hyb}^{k-1}(b)$ perfectly.

– If $y_0, y_{m+1}, \cdots, y_\ell$ are chosen randomly, then $\bar{z}_0, \bar{z}_1, \cdots, \bar{z}_t$ defines a random polynomial $\hat{f}$ with constraint that for all $i \in C$, $\hat{f}(i) = f^\star(i)$. Therefore, in this case, $\mathcal{B}$ simulates $\mathsf{Hyb}^k(b)$ perfectly.

Since $\eta^\star = \eta$ happens with probability $1/q_{H_1}$, if $\mathcal{A}$ distinguishes hybrids $\mathsf{Hyb}^{k-1}(b)$ and $\mathsf{Hyb}^k(b)$ with a non-negligible probability δ, then $\mathcal{B}$ can break the (extended) DDH assumption with a non-negligible probability δ/q_{H_1}.

Now we are left to show that $\mathsf{Hyb}^{q_E}_{\mathcal{A}}(0)$ is indistinguishable from $\mathsf{Hyb}^{q_E}_{\mathcal{A}}(1)$. For a j-th distinct x, $\mathsf{pc}(x, \cdot, \cdot)$ is defined using a unique random t-degree polynomial $f^j(x)$ which only matches with $f(x)$ on C. Since an adversary is allowed to make at most $t - |C|$ evaluation queries on $x^\star$, the adversary can learn the value of $f^j(x)$ on at most t points. As a result, the product $\prod_{i \in V} z_i^{\star \lambda_{0,i,V}}$ with $|V| > t$ when $b = 0$ has at least one $z_i^\star$ for which adversary has no information. Thus, the product appears random to the adversary, which means $\mathsf{Hyb}^{q_E}_{\mathcal{A}}(0)$ and $\mathsf{Hyb}^{q_E}_{\mathcal{A}}(1)$ are indistinguishable.

C Security proofs for Dfinity-DVRF

Theorem 4. $\mathcal{V}^{\mathsf{Dfinity\text{-}DVRF}}$ satisfies standard pseudorandomness under co-CDH assumption in the random oracle model.

Proof. Recall that standard pseudorandomness does not allow the adversary to query partial evaluations on the challenge plaintext $x^\star$.

Similar to Theorem 2, we construct $\mathsf{Hyb}^{sim}(b)$ to simulate the DKG protocol and the original standard pseudorandomness game is indistinguishable with $\mathsf{Hyb}^{sim}(b)$ because of the secrecy of the simulator. Below we shall prove $\mathsf{Hyb}^{sim}(0)$ and $\mathsf{Hyb}^{sim}(1)$ are indistinguishable under co-CDH assumption. Suppose there exists an adversary $\mathcal{A}$ that distinguishes $\mathsf{Hyb}^{sim}(0)$ and $\mathsf{Hyb}^{sim}(1)$, 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 $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, $\alpha, \beta \xleftarrow{R} \mathbb{Z}_q$. $\mathcal{B}$'s goal is to output $g_1^{\alpha\beta}$:

1. Give the public parameters $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2)$ and a list of servers $S = \{1, 2, \ldots, \ell\}$ to $\mathcal{A}$.
2. $\mathcal{A}$ chooses a set C of servers with $|C| \leq t$ to corrupt and send C to $\mathcal{B}$. W.l.o.g., assume $C = \{1, 2, \ldots, m\}$.
3. The random oracle H_1 is answered as follows: initialise $\mathcal{L}_{H_1} = \emptyset$. Let q_{H_1} be the total number of distinct random oracle queries asked in this game. Choose an index $\eta^\star \xleftarrow{R} [q_{H_1}]$ uniformly at random.
   – If there exists a tuple $(x, r, h) \in \mathcal{L}_{H_1}$, output h.
   – Otherwise,
     • if this is the $\eta^\star$-th distinct call, set $r = \perp$ and $h = g_1^\beta$ where g_1^β is from the co-CDH problem.
     • else choose a random $r \xleftarrow{R} \mathbb{Z}_q$ and set $h = g_1^r$.
     • Update $\mathcal{L}_{H_1} = \mathcal{L}_{H_1} \cup (x, r, h)$ and output h.

   Give random oracle access to $\mathcal{A}$.
4. The random oracle H_2 is programmed as follows: Define a list $\mathcal{L}_{H_2} = \emptyset$. For a query on y,
   – If there exists a tuple $(y, c, *)$, then output c.
   – Else verify if $e(y, g_2) = e(g_1^\beta, g_2^\alpha)$:
     • If true, check if there exists a tuple $(\perp, c, \mathsf{true})$ (set by the challenge query) and update the list by changing the $\perp$ to y. If such a tuple does not exist, then choose a random $c \xleftarrow{R} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup (y, c, \mathsf{true})$. The oracle outputs c.
     • If false, choose a random $c \xleftarrow{R} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup (y, c, \mathsf{false})$. The oracle outputs c.

– **Generating phase:**

1. Each server S_i performs a Pedersen-VSS of a random value $a_{i,0}$:
   (a) S_i chooses two random t-degree polynomials f_i, f'_i over $\mathbb{Z}_q$:

$$f_i(z) = a_{i,0} + a_{i,1}z + \cdots + a_{i,t}z^t \qquad f'_i(z) = b_{i,0} + b_{i,1}z + \cdots + b_{i,t}z^t$$

   S_i broadcasts $C_{i,k} = g_2^{a_{i,k}} h_2^{b_{i,k}}$ for $0 \leq k \leq t$. S_i computes the shares $s_{i,j} = f_i(j), s'_{i,j} = f'_i(j)$ for $1 \leq j \leq \ell$ and sends $s_{i,j}, s'_{i,j}$ to server S_j.
   (b) Each server S_j verifies the shares he received from the other servers. S_j checks if

$$g_2^{s_{i,j}} h_2^{s'_{i,j}} = \prod_{k=0}^{t} (C_{i,k})^{j^k} \tag{3}$$

   for $1 \leq i \leq \ell$. If the check fails for an index i, S_j broadcasts a complaint against S_i.
   (c) Each server S_i who received a complaint from server S_j broadcasts the values $s_{i,j}, s'_{i,j}$ that satisfy Equation 5.
   (d) Each server marks as *disqualified* any server that either
   - Received more than t complaints in Step 1b, or
   - Answered to a complaint in Step 1c with values that falsify Equation 5.
2. Each server then builds the set of non-disqualified players QUAL.
3. Each server S_i sets his share of the secret as $\mathsf{sk}_i = \sum_{j \in \mathrm{QUAL}} s_{j,i}$, the verification key $\mathsf{vk}_i = g_2^{\mathsf{sk}_i}$ and the value $r_i = \sum_{j \in \mathrm{QUAL}} s'_{j,i}$.

– **Extracting phase:**

4. Each server $i \in$ QUAL exposes $A_{i,0} = g_2^{a_{i,0}}$ via **Feldman-VSS**
   (a) Each server S_i with $i \in$ QUAL, broadcasts $A_{i,k} = g_2^{a_{i,k}}$ for $0 \leq k \leq t$.
   (b) Each server S_j verifies the values broadcast by the other servers in QUAL, formally, for each $i \in$ QUAL, S_j checks if

$$g_2^{s_{i,j}} = \prod_{k=0}^{t} (A_{i,k})^{j^k} \tag{4}$$

   If the check fails for some index i, S_j complains against S_i by broadcasting the values $s_{i,j}, s'_{i,j}$ that satisfy Equation 5 but not Equation 6.
   (c) For server S_i who receives at least one valid complaint, i.e., values which satisfy Equation 5 but not Equation 6, the other servers run the re-construction phase of **Pedersen-VSS** to compute $a_{i,0}, f_i, A_{i,k}$ for $0 \leq k \leq t$. Compute the global public key as $\mathsf{pk} = \prod_{i \in \mathrm{QUAL}} A_{i,0}$. The distributed secret value $\mathsf{sk} = \sum_{i \in \mathrm{QUAL}} a_{i,0}$ is not explicitly computed by any party.
   (d) Each server S_i can reconstruct the verification keys of other servers:
   - Compute $v_k = \prod_{i \in \mathrm{QUAL}} A_{i,k}$ for $0 \leq k \leq t$.
   - For each $j \in$ QUAL such that $j \neq i$, compute $\mathsf{vk}_j = \prod_{k=0}^{t} v_k^{j^k}$.

Fig. 2: Adaptation of the DKG protocol [GJKR07] to pairing groups for Dfinity-DVRF.

Give random oracle access to $\mathcal{A}$. Note that, when the pairing verification is successful, we can derive that $y = g_1^{\alpha\beta}$.

5. Run the simulator of the secure DKG protocol to interact with the corrupted servers. The protocol constructs $C_{i,k}$ using g_2, h_2 from $\mathbb{G}_2$. Let QUAL be the non-disqualified servers. Assume the combined polynomial after the generating phase is f. In the extracting phase, when constructing the ℓ-th polynomial $f_\ell^\star$ to replace f_ℓ, the simulator sets $f_\ell^\star$ to have the values $\alpha - \sum_{j \in \mathrm{QUAL} \setminus \{\ell\}} f_j(0)$, $f_\ell(1), \ldots, f_\ell(t)$. The computation of $A_{i,j}, A^\star_{\ell,j}$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $0 \le j \le t$ are similar to $\mathsf{Hyb}^{sim}_{\mathcal{A}}(b)$ in Theorem 2 except they are constructed on $\mathbb{G}_2$. The global public key is $\mathsf{pk} = g_2^\alpha$. The corrupted servers that are included in QUAL have the shares $\mathsf{sk}_i = f(i)$ and $\mathsf{vk}_i = g_2^{f(i)}$. The shares of the honest servers $i \in [t+1, \ell]$ are set to be $\mathsf{sk}_i = f(i)$ and $\mathsf{vk}_i = g_2^{f(i)}$. Note that, the simulator cannot compute sk_i for the honest servers $i \in [t+1, \ell]$ because it does not know the value of α, but can compute vk_i as $g_2^{\alpha\lambda_{i,0,T}} \prod_{j \in T, j \neq 0} g_2^{f(j)\lambda_{i,j,T}}$ with $T = \{0, 1, \ldots, t\}$.
6. On an evaluation query (Eval, x, i) for an honest i, invoke random oracle H_1 to get $H_1(x) = (x, r, h)$ and
   (a) return vk_i^r if $r \neq \perp$
   (b) return $\perp$ if $r = \perp$
7. On the challenge query $(\mathsf{Challenge}, x^\star, \{(i, z_i^\star)\}_{i \in U}, V)$, where $|V| > t$ and $V \subseteq$ QUAL and $U \subseteq V \cap C$ and $z_i^\star$ a set of evaluation shares from the corrupted servers, is answered as follows:
   (a) If $x^\star$ was not the $\eta^\star$-the query to H_1, then $\mathcal{B}$ aborts.
   (b) Otherwise do as follows:
      i. Check if $e(v_i, g_2) = e(H_1(x), \mathsf{vk}_i)$ for $i \in U$. If any check fails, output 0 and stop.
      ii. If there exists a tuple (y, c, true) in H_2, then set $v^\star = c$. Otherwise choose $v^\star \xleftarrow{R} \mathbb{Z}_q$ and update $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup \{\perp, v^\star, \mathsf{true}\}$. Depending on b do as follows:
         A. If $b = 0$ then return $v^\star$;
         B. Else return a uniform random.
8. Continue answering evaluation queries as before, but if $\mathcal{A}$ makes queries of the form $(\mathsf{Eval}, x^\star, \cdot)$ then output 0 and stop.
9. Receive a guess b' from $\mathcal{A}$.
10. If there exists a tuple (y, c, true) with $y \neq \perp$ in H_2, $\mathcal{B}$ outputs y as a solution to the co-CDH problem.

The probability that $\mathcal{B}$ does not abort is $1/q_{H_1}$ since $\eta^\star$ is uniformly and randomly chosen. Let's consider the case when `abort` does not happen. In this case, the challenge plaintext $x^\star$ is also the $\eta^\star$-th distinct query to H_1 and $H_1(x^\star) = g_1^\beta$. The adversary is not allowed to query $(\mathsf{Eval}, x^\star, \cdot)$ due to the definition of standard pseudorandomness. In other words, the Step 6b will never be executed. In this case $\mathcal{B}$ simulates the standard pseudorandomness game perfectly from $\mathcal{A}$'s point of view.

We define an event C as when the adversary queries $y^\star$ to H_2 such that $e(y^\star, g_2) = e(g_1^\beta, g_2^\alpha)$, i.e., $y^\star = g_1^{\alpha\beta}$. We shall argue that the probability that C does not happen is the same for $b = 0$ and $b = 1$, i.e., $\Pr[\neg C | b = 0] = \Pr[\neg C | b = 1]$. Before the challenge query, there is no information about b, the adversary $\mathcal{A}$ sees exactly the same probability distribution on $\mathcal{B}$'s outputs no matter $b = 0$ or $b = 1$. Thus the probability that C does not happen before the challenge query is the same. After the challenge query, because we assume C has not occurred so far, the challenge query returns a uniform random that has never been used before in both $b = 0$ and $b = 1$. Thus, before the adversary queries the random oracle H_2 on $y^\star$, the adversary sees the same probability distribution of $\mathcal{B}$'s outputs after the challenge query. This gives us $\Pr[\neg C | b = 0] = \Pr[\neg C | b = 1]$. We can also derive that $\Pr[C | b = 0] = \Pr[C | b = 1]$ and $\Pr[C] = \Pr[C | b = 0] \Pr[b = 0] + \Pr[C | b = 1] \Pr[b = 1] = \Pr[C | b = 0]$. Moreover, when C does not happen, the adversary sees the same probability distribution on $\mathcal{B}$'s outputs, and thus the probability that $\mathcal{A}$ outputs 1 is also the same for $b = 0$ and $b = 1$, i.e., $\Pr[b' =$

$1|b=0,\neg C] = \Pr[b'=1|b=1,\neg C]$. Therefore we can compute $\mathcal{A}$'s advantage of distinguishing $b=0$ and $b=1$ as

$$\begin{aligned}
\mathbf{Adv} &= |\Pr[b'=1|b=0] - \Pr[b'=1|b=1]| \\
&= |\Pr[b'=1|b=0,C]\Pr[C|b=0] + (\Pr[b'=1|b=0,\neg C]\Pr[\neg C|b=0] \\
&\quad - \Pr[b'=1|b=1,C]\Pr[C|b=1] - \Pr[b'=1|b=1,\neg C]\Pr[\neg C|b=1]| \\
&= |\Pr[b'=1|b=0,C]\Pr[C|b=0] - \Pr[b'=1|b=1,C]\Pr[C|b=1]| \\
&= |\Pr[b'=1|b=0,C] - \Pr[b'=1|b=1,C]| \cdot \Pr[C|b=0] \\
&\leq \Pr[C|b=0] = \Pr[C]
\end{aligned}$$

Since we assume $\mathcal{A}$ distinguishes $b=0$ and $b=1$ with non-negligible advantage $\mathbf{Adv}$, C happens with non-negligible probability $\Pr[C]$. Therefore $\mathcal{B}$ outputs $y^\star$ as a solution to the co-CDH problem with non-negligible probability $\Pr[C]/q_{H_1}$.

D Security proofs for GLOW-DVRF

Theorem 6. $\mathcal{V}^{\mathsf{GLOW\text{-}DVRF}}$ is strongly pseudorandom under the *XDH* assumption and *co-CDH* assumption in the random oracle model.

Proof. Similar to Theorem 2, we construct $\mathsf{Hyb}^{zk}(b)$ to simulate all the NIZKs using random oracle H_3 and construct $\mathsf{Hyb}^{sim}(b)$ to simulate the DKG protocol. We have that the original strong pseudorandomness game and $\mathsf{Hyb}^{zk}(b)$ are indistinguishable due to the zero-knowledge property of NIZKs, and $\mathsf{Hyb}^{zk}(b)$ and $\mathsf{Hyb}^{sim}(b)$ are indistinguishable due to the secrecy of the simulator. Assume m is the total number of corrupted servers, we consider two cases $m=t$ and $m<t$.

The case when $m=t$: this is the simplest case and the proof is similar to Theorem 4. This is because $m=t$ means an adversary has already compromised t servers and cannot issue any evaluation query on the challenge plaintext $x^\star$. Given an adversary $\mathcal{A}$ that distinguishes $\mathsf{Hyb}^{sim}(0)$ and $\mathsf{Hyb}^{sim}(1)$, the construction of an adversary $\mathcal{B}$ breaks the co-CDH assumption using $\mathcal{A}$ as a subroutine is exactly the same as the proof of Theorem 4 except that:

- An additional random oracle H_3 is programmed as follows: Define a list $\mathcal{L}_{H_3} = \emptyset$. For a query on y, if there exists a tuple (y,c), then output c. Otherwise choose a random $c \xleftarrow{R} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_3} = \mathcal{L}_{H_3} \cup (y,c)$. The oracle outputs c. Give random oracle access to $\mathcal{A}$.
- Step 5 is replaced as follows: Run the simulator of the secure DKG protocol to interact with the corrupted servers. Let QUAL be the non-disqualified servers. Assume the combined polynomial after the generating phase is f. In the extracting phase, when constructing the ℓ-th polynomial $f_\ell^\star$ to replace f_ℓ, the simulator sets up the points $\alpha - \sum_{j\in\mathrm{QUAL}\setminus\{\ell\}} f_j(0)$, $f_\ell(1)$, $\ldots$, $f_\ell(t)$ for $f_\ell^\star$. The $A_{i,j}, A^\star_{\ell,j}$ and $B^\star_{i,0}$ for $i \in \mathrm{QUAL}\setminus\{\ell\}$ and $0 \leq j \leq t$ can be easily computed as $A_{i,j}, A^\star_{\ell,j}$ for $i \in \mathrm{QUAL}\setminus\{\ell\}$ and $0 \leq j \leq t$ can be computed similarly as in $\mathsf{Hyb}^{sim}_{\mathcal{A}}(b)$ in Theorem 2. $B^\star_{i,0}$ for $i \in \mathrm{QUAL}\setminus\{\ell\}$ and $B^\star_{\ell,0}$ are computed as:
  - For $i \in \mathrm{QUAL}\setminus\{\ell\}$, $B_{i,0} = g_2^{a_{i,0}}$ since $\mathcal{B}$ has the coefficients of all the polynomials except the ℓ-th polynomial.
  - $B_{\ell,0} = g_2^\alpha \prod_{i\in\mathrm{QUAL}\setminus\{\ell\}} B_{i,0}^{-1}$.

  The global public key $\mathsf{pk} = \prod_{i\in\mathrm{QUAL}} B_{i,0} = g_2^\alpha$. The corrupted servers that are included in QUAL have the shares $\mathsf{sk}_i = f(i)$ and $\mathsf{vk}_i = g_1^{f(i)}$ for $i \in \mathrm{QUAL} \cap C$. The shares of the honest servers are set to be $\mathsf{sk}_i = f(i)$ and $\mathsf{vk}_i = g_1^{f(i)}$ for $i \in [m+1, \ell]$. Note that, the simulator cannot compute sk_i for the honest servers $i \in [t+1, \ell]$ because it does not know the value of α, but can compute $\mathsf{vk}_i = g_1^{\alpha\lambda_{i,0,T}} \prod_{j\in T, j\neq 0} g_1^{f(j)\lambda_{i,j,T}}$ with $T = \{0,1,\ldots,t\}$.

– **Generating phase:**

1. Each server S_i performs a Pedersen-VSS of a random value $a_{i,0}$:
   (a) S_i chooses two random t-degree polynomials f_i, f'_i over $\mathbb{Z}_q$:

$$f_i(z) = a_{i,0} + a_{i,1}z + \cdots + a_{i,t}z^t \qquad f'_i(z) = b_{i,0} + b_{i,1}z + \cdots + b_{i,t}z^t$$

   S_i broadcasts $C_{i,k} = g_1^{a_{i,k}} h_1^{b_{i,k}}$ for $0 \leq k \leq t$. S_i computes the shares $s_{i,j} = f_i(j), s'_{i,j} = f'_i(j)$ for $1 \leq j \leq \ell$ and sends $s_{i,j}, s'_{i,j}$ to server S_j.
   (b) Each server S_j verifies the shares he received from the other servers. S_j checks if

$$g_1^{s_{i,j}} h_1^{s'_{i,j}} = \prod_{k=0}^{t} (C_{i,k})^{j^k} \tag{5}$$

   for $1 \leq i \leq \ell$. If the check fails for an index i, S_j broadcasts a complaint against S_i.
   (c) Each server S_i who received a complaint from server S_j broadcasts the values $s_{i,j}, s'_{i,j}$ that satisfy Equation 5.
   (d) Each server marks as *disqualified* any server that either
   - Received more than t complaints in Step 1b, or
   - Answered to a complaint in Step 1c with values that falsify Equation 5.
2. Each server then builds the set of non-disqualified players QUAL.
3. Each server S_i sets his share of the secret as $\mathsf{sk}_i = \sum_{j \in \mathrm{QUAL}} s_{j,i}$, the verification key $\mathsf{vk}_i = g_1^{\mathsf{sk}_i}$ and the value $r_i = \sum_{j \in \mathrm{QUAL}} s'_{j,i}$.

– **Extracting phase:**

4. Each server $i \in$ QUAL exposes exposes $A_{i,0} = g_1^{a_{i,0}}$ via **Feldman-VSS** and $B_{i,0} = g_2^{a_{i,0}}$
   (a) Each server S_i with $i \in$ QUAL, broadcasts broadcasts $A_{i,k} = g_1^{a_{i,k}}$ for $0 \leq k \leq t$ and $B_{i,0} = g_2^{a_{i,0}}$.
   (b) Each server S_j verifies the values broadcast by the other servers in QUAL, formally, for each $i \in$ QUAL, S_j checks if

$$g_1^{s_{i,j}} = \prod_{k=0}^{t} (A_{i,k})^{j^k} \text{ and } e(A_{i,0}, g_2) = e(g_1, B_{i,0}) \tag{6}$$

   If any of the checks fails for some index i, S_j complains against S_i by broadcasting the values $s_{i,j}, s'_{i,j}$ or $B_{i,0}$ that satisfy Equation 5 but not Equation 6.
   (c) For server S_i who receives at least one valid complaint, i.e., values which satisfy Equation 5 but not Equation 6, the other servers run the re-construction phase of **Pedersen-VSS** to compute $a_{i,0}, f_i, A_{i,k}$ for $0 \leq k \leq t$. Compute the global public key as $\mathsf{pk} = \prod_{i \in \mathrm{QUAL}} B_{i,0}$. The distributed secret value $\mathsf{sk} = \sum_{i \in \mathrm{QUAL}} a_{i,0}$ is not explicitly computed by any party.
   (d) Each server S_i can reconstruct the verification keys of other servers:
   - Compute $v_k = \prod_{i \in \mathrm{QUAL}} A_{i,k}$ for $0 \leq k \leq t$.
   - For each $j \in$ QUAL such that $j \neq i$, compute $\mathsf{vk}_j = \prod_{k=0}^{t} v_k^{j^k}$.

Fig. 3: DKG protocol for GLOW-DVRF.

- Step 6 is replaced as follows: On an evaluation query (Eval, x, i) for an honest i, invoke random oracle H_1 to get $(x, r, h) \in \mathcal{L}_{H_1}$ and
  1. return (vk_i^r, π_i) if $r \neq \perp$, where π_i is a simulated proof generated by calling the random oracle H_3
  2. return $\perp$ if $r = \perp$
- Step 7 is replaced as follows: On the challenge query $(\mathsf{Challenge}, x^\star, \{(i, z_i^\star, \pi_i)\}_{i \in U}, V)$, where $|V| > t$ and $V \subseteq \mathrm{QUAL}$ and $U \subseteq V \cap C$ and $z_i^\star$ a set of evaluation shares from the corrupted servers, is answered as follows:
  1. if $x^\star$ was not the $\eta^\star$-th query to H_1, then $\mathcal{B}$ aborts.
  2. Otherwise do as follows:
     (a) Run $\mathsf{VerifyEq}_{H_3}$ to check the proofs π_i for $i \in U$. If any check fails, output 0 and stop.
     (b) If there exists a tuple (y, c, true) in H_2, then set $v^\star = c$. Otherwise choose $v^\star \xleftarrow{R} \mathbb{Z}_q$ and update $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup \{\perp, v^\star, \mathsf{true}\}$. Depending on b do as follows:
        i. If $b = 0$ then return $v^\star$;
        ii. Else return a uniform random.

The rest of the analysis is similar to Theorem 4. The probability that $\mathcal{B}$ does not abort is $1/q_{H_1}$ since $\eta^\star$ is uniformly and randomly chosen. Let's assume `abort` does not happen. We define an event C as when the adversary queries $y^\star$ to H_2 such that $e(y^\star, g_2) = e(g_1^\beta, g_2^\alpha)$, i.e., $y^\star = g_1^{\alpha\beta}$. We can prove that $\mathcal{A}$'s advantage $\mathbf{Adv}$ of distinguishing $b = 0$ and $b = 1$ is not bigger than $\Pr[C]$, i.e., $\mathbf{Adv} \leq \Pr[C]$. Since we assume $\mathcal{A}$ distinguishes $b = 0$ and $b = 1$ with non-negligible advantage $\mathbf{Adv}$, C happens with non-negligible probability $\Pr[C]$. Therefore $\mathcal{B}$ outputs $y^\star$ as a solution to the co-CDH problem with non-negligible probability $\Pr[C]/q_{H_1}$.

The case when $m < t$: in this case, the adversary is allowed to make up to $t - m(\geq 1)$ evaluation queries on the challenge plaintext $x^\star$. To prove the strong pseudorandomness, we shall construct k-hybrids $\mathsf{Hyb}_{\mathcal{A}}^k(b)$ with $0 \leq k \leq q_E$ and q_E the total number of distinct x in the evaluation/challenge queries. We will show that

- $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^k(b)$ for $1 \leq k \leq q_E$ are indistinguishable under XDH assumption
- $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(0)$ and $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(1)$ are indistinguishable under co-CDH assumption

Definition of the k-hybrids. The hybrid games $\mathsf{Hyb}_{\mathcal{A}}^k(b)$ are exactly the same as $\mathsf{Hyb}_{\mathcal{A}}^{sim}(b)$ except how the evaluation/challenge queries are answered on the first k distinct x:

- After the simulator runs the secure DKG protocol to set up the system, the simulator in addition chooses k t-degree random polynomials $f^1, f^2, \cdots, f^k$ such that $f^j(0) = \mathsf{sk}$ and for all $i \in C$, $f^j(i) = \mathsf{sk}_i$. The reason $f^j(0) = \mathsf{sk}$ is because the adversary has the global public key $\mathsf{pk} = g_2^{\mathsf{sk}}$ and can test if $z = \mathsf{sk}$ in an evaluation $H(x)^z$ for any x by checking the equality of pairings $e(H(x)^z, g_2) = e(H(x), \mathsf{pk})$.
- The evaluation queries (Eval, x, i) are answered similar to Theorem 2 by calling the function $\mathsf{pc}(x, j, i)$.
- The challenge query $(\mathsf{Challenge}, x^\star, \{(i, z_i^\star, \pi_i)\}_{i \in U}, V)$ is answered the same as Theorem 2 except Step 3c is replaced by: Compute $z = \prod_{i \in V} {z_i^\star}^{\lambda_{0,i,V}}$, query the random oracle H_2 on z and set $v^\star = H_2(z)$.

Obviously, $\mathsf{Hyb}_{\mathcal{A}}^0(b)$ is identical to $\mathsf{Hyb}_{\mathcal{A}}^{sim}(b)$. $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^k(b)$ only differs in the evaluation queries for the k-th distinct x. Below we shall prove that $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^k(b)$ are indistinguishable from the attacker's point of view.

Lemma 2. *For $1 \leq k \leq q_E$, $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^k(b)$ are indistinguishable under XDH assumption in the random oracle model.*

Proof. We show that if there exists a PPT adversary $\mathcal{A}$ that can distinguish between the hybrids $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$ with non-negligible probability then we can construct a PPT adversary $\mathcal{B}$ to break an extended version of the DDH assumption [AMMR18] using $\mathcal{A}$ as a subroutine.

The extended XDH problem is given by

$$(\mathbb{G}, q, g_1, g_2, g_1^{\alpha_1}, \cdots, g_1^{\alpha_w}, g_1^{\beta}, y_1, \cdots, y_w)$$

where $y_i = g_1^{\alpha_i \beta}$ for all $i \in \{1, \ldots, w\}$ or randoms. We now construct $\mathcal{B}$ using $\mathcal{A}$ as follows:

1. Give the public parameters $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2)$ and a list of servers $S = \{1, 2, \ldots, \ell\}$ to $\mathcal{A}$.
2. $\mathcal{A}$ chooses a set $C = \{1, 2, \ldots, m\}$ of servers with $m < t$ to corrupt and send C to $\mathcal{B}$.
3. The random oracle H_1 is answered as follows: initialise $\mathcal{L}_{H_1} = \emptyset$. Let q_{H_1} be the total number of distinct random oracle queries asked in this game. Choose an index $\eta^\star \xleftarrow{R} [q_{H_1}]$ uniformly at random.
   - If there exists a tuple $(x, r, h) \in \mathcal{L}_{H_1}$, output h.
   - Otherwise,
     - if this is the $\eta^\star$-th distinct call, set $r = \perp$ and $h = g_1^{\beta}$ where g_1^{β} is from the co-CDH problem.
     - else choose a random $r \xleftarrow{R} \mathbb{Z}_q$ and set $h = g_1^r$.
   - Update $\mathcal{L}_{H_1} = \mathcal{L}_{H_1} \cup (x, r, h)$ and output h.

   Give random oracle access to $\mathcal{A}$.
4. The random oracle H_2 is programmed as follows: Define a list $\mathcal{L}_{H_2} = \emptyset$. For a query on y, if there exists a tuple (y, c), then output c. Otherwise choose a random $c \xleftarrow{R} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup (y, c)$. The oracle outputs c. Give random oracle access to $\mathcal{A}$.
5. The random oracle H_3 is programmed as follows: Define a list $\mathcal{L}_{H_3} = \emptyset$. For a query on y, if there exists a tuple (y, c), then output c. Otherwise choose a random $c \xleftarrow{R} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_3} = \mathcal{L}_{H_3} \cup (y, c)$. The oracle outputs c. Give random oracle access to $\mathcal{A}$.
6. Run the simulator of the secure DKG protocol to interact with the corrupted servers. Let QUAL be the non-disqualified servers. Assume the combined polynomial after the generating phase is f. The simulator chooses $\gamma \xleftarrow{R} \mathbb{Z}_q$. In the extracting phase, the ℓ-th polynomial $f_\ell^\star$ is constructed using the points $\gamma - \sum_{j \in \mathrm{QUAL} \setminus \{\ell\}} f_j(0)$, $f_\ell(1)$, $\ldots$, $f_\ell(m)$, $\alpha_{m+1} - \sum_{j \in \mathrm{QUAL} \setminus \{\ell\}} f_j(m+1)$, $\ldots$, $\alpha_t - \sum_{j \in \mathrm{QUAL} \setminus \{\ell\}} f_j(t)$ where $\alpha_{m+1}, \ldots, \alpha_t$ are from the XDH problem. $A_{i,j}, A_{\ell,j}^\star$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $0 \le j \le t$ can be computed similar to Lemma 1. $B_{i,0}^\star$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $B_{\ell,0}^\star$ are computed as:
   - For $i \in \mathrm{QUAL} \setminus \{\ell\}$, $B_{i,0} = g_2^{a_{i,0}}$ since $\mathcal{B}$ has the coefficients of all the polynomials except the ℓ-th polynomial.
   - $B_{\ell,0}^\star = g_2^{\gamma} \prod_{i \in \mathrm{QUAL} \setminus \{\ell\}} B_{i,0}^{-1}$.

   Let the final combined polynomial be $f^\star$. We know that $f^\star(0) = \gamma$, $f^\star(1) = f(1), \ldots, f^\star(m) = f(m)$, $f^\star(m+1) = \alpha_{m+1}, \ldots, f^\star(t) = \alpha_t$. The global public key is $\mathsf{pk} = g_2^{\gamma}$. The shares of the honest servers are set to be $\mathsf{sk}_i = \alpha_i$ and the corresponding verification keys are $\mathsf{vk}_i = g_1^{\alpha_i}$ for $i \in [m+1, t]$ despite $\alpha_{m+1}, \ldots, \alpha_t$ are unknown. For the honest servers $i \in [t+1, \ell]$, we cannot compute their shares sk_i because $\alpha_{m+1}, \ldots, \alpha_t$ are unknown, but we can compute their verification keys as $\mathsf{vk}_i = g_1^{\gamma \lambda_{i,0,T}} \prod_{j \in T, j \neq 0} g_1^{f^\star(j) \lambda_{i,j,T}}$ with $T = \{0, \ldots, t\}$.
7. Choose $k-1$ t-degree random polynomials, $f^1, f^2, \ldots, f^{k-1}$ such that $f^j(0) = \gamma$ and for all $i \in C$ and $1 \le j \le k-1$, $f^j(i) = f^\star(i)$.
8. Compute $\bar{z}_0 = g_1^{\beta\gamma}$ and $\bar{z}_i = (g_1^{\beta})^{f^\star(i)}$ for $i \in C$. Set $\bar{z}_i = y_i$ for $m+1 \le i \le t$ where $y_{m+1}, \ldots, y_t$ are from the XDH problem. Then for all $m+1 \le i \le \ell$, compute $\bar{z}_i = \prod_{j \in T} \bar{z}_j^{\lambda_{i,j,T}}$ with $T = \{0, 1, \ldots, t\}$.

9. Define a function $\mathsf{pc}(x, i, j)$ as follows: invoke random oracle H_1 to get $(x, r, h) \in \mathcal{L}_{H_1}$ and
   - return $(g_1^{f^j(i)\cdot r}, \pi_i)$ if $j < k$
   - return $(\bar{z}_i, \pi_i)$ if $j = k$
   - return (vk_i^r, π_i) if $j > k$ (if $r = \perp$, return $\perp$)

   where π_i is a simulated proof generated by calling the random oracle H_3.
10. On an evaluation query (Eval, x, i) for an honest i, if x is the j-th distinct value, then return $\mathsf{pc}(x, j, i)$.
11. On the challenge query $(\mathsf{Challenge}, x^\star, \{(i, z_i^\star, \pi_i)\}_{i\in U}, V)$, where $|V| > t$ and $V \subseteq$ QUAL and $U \subseteq V \cap C$ and $z_i^\star$ a set of evaluation shares from the corrupted servers, is answered as follows:
    (a) if $x^\star$ was queried in the evaluation phase and it was the j-th distinct value, then let $j^\star = j$. Else, assume there are so far j' distinct evaluation queries and let $j^\star = j' + 1$.
    (b) If $\mathcal{A}$ has made at least $t + 1 - m$ queries of the form $(\mathsf{Eval}, x^\star, \cdot)$, then output 0 and stop.
    (c) Otherwise do as follows:
        i. Run $\mathsf{VerifyEq}_{H_3}$ to check the proofs π_i for $i \in U$. If any check fails, output 0 and stop.
        ii. Set $(z_i^\star, \cdot) = \mathsf{pc}(x^\star, j^\star, i)$ for $i \in V \setminus C$.
        iii. Compute $z = \prod_{i\in V} z_i^{\star\lambda_{0,i,V}}$ and query random oracle H_2 on z. Let $v^\star = H_2(z)$. Depending on b do as follows:
            A. If $b = 0$ then return $v^\star$;
            B. Else return a uniform random.
12. Continue answering evaluation queries as before, but if $\mathcal{A}$ makes queries of the form $(\mathsf{Eval}, x^\star, i)$ for some $i \in$ QUAL $\setminus C$ and i is the $(t + 1 - m)$-th honest server that $\mathcal{A}$ contacted then output 0 and stop.
13. Receive a guess b' from $\mathcal{A}$ and output b'.

Suppose the k-th distinct evaluation query x_k is the η-th query for random oracle H_1. Let's consider the case when $\eta^\star = \eta$. pc in Step 9 never returns $\perp$ because r is set to $\perp$ only for the $\eta^\star$-th call and we have assumed that this call is for x_k.

- If $y_{m+1} = g^{\alpha_{m+1}\beta}, \ldots, y_\ell = g^{\alpha_\ell\beta}$, pc returns $\bar{z}_i$ on x_k which is equal to $H_1(x_k)^{f^\star(i)}$. In this case, $\mathcal{B}$ simulates $\mathsf{Hyb}^{k-1}(b)$ perfectly.
- If $y_{m+1}, \cdots, y_\ell$ are chosen randomly, then $\bar{z}_1, \cdots, \bar{z}_t$ defines a random polynomial $\hat{f}$ with constraint that $\hat{f}(0) = \gamma$ and for all $i \in C$, $\hat{f}(i) = f^\star(i)$. Therefore, in this case, $\mathcal{B}$ simulates $\mathsf{Hyb}^k(b)$ perfectly.

Since $\eta^\star = \eta$ happens with probability $1/q_{H_1}$, if $\mathcal{A}$ distinguishes hybrids $\mathsf{Hyb}^{k-1}(b)$ and $\mathsf{Hyb}^k(b)$ with a non-negligible probability δ, then $\mathcal{B}$ can break the (extended) XDH assumption with a non-negligible probability δ/q_{H_1}.

Now we are left to show that $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(0)$ is indistinguishable from $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(1)$. In $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(b)$, for j-th distinct x, $\mathsf{pc}(x, j, \cdot)$ is computed using a unique random t-degree polynomial f^j which only matches with $f^\star$ on $C \cup \{0\}$.

Lemma 3. $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(0)$ *and* $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(1)$ *are indistinguishable under co-CDH assumption in the random oracle model.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that distinguishes $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(0)$ and $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(1)$, 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 $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, $\alpha, \beta \xleftarrow{R} \mathbb{Z}_q$. $\mathcal{B}$'s goal is to output $g_1^{\alpha\beta}$:

1. Give the public parameters $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2)$ and a list of servers $S = \{1, 2, \ldots, \ell\}$ to $\mathcal{A}$.
2. $\mathcal{A}$ chooses a set $C = \{1, 2, \ldots, m\}$ of servers with $m < t$ to corrupt and send C to $\mathcal{B}$.

3. The random oracle H_1 is answered as follows: initialise $\mathcal{L}_{H_1} = \emptyset$. Let q_{H_1} be the total number of distinct random oracle queries asked in this game. Choose an index $\eta^\star \xleftarrow{R} [q_{H_1}]$ uniformly at random.
   - If there exists a tuple $(x, r, h) \in \mathcal{L}_{H_1}$, output h.
   - Otherwise,
     - If this is the $\eta^\star$-th distinct call, set $r = \perp$ and $h = g_1^\beta$ where g_1^β is from the co-CDH problem.
     - Else choose a random $r \xleftarrow{R} \mathbb{Z}_q$ and set $h = g_1^r$.
   - Update $\mathcal{L}_{H_1} = \mathcal{L}_{H_1} \cup \{(x, r, h)\}$ and output h.

   Give random oracle access to $\mathcal{A}$.
4. The random oracle H_2 is programmed as follows: Define a list $\mathcal{L}_{H_2} = \emptyset$. For a query on y,
   - If there exists a tuple $(y, c, \cdot)$, then output c.
   - Else verify if $e(y, g_2) = e(g_1^\beta, g_2^\alpha)$:
     - If true, check if there exists a tuple $(\perp, c, \mathsf{true})$ (set by the challenge query) and update the list by changing the $\perp$ to y. If such a tuple does not exist, then choose a random $c \xleftarrow{R} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup (y, c, \mathsf{true})$. The oracle outputs c.
     - If false, choose a random $c \xleftarrow{R} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup \{(y, c, \mathsf{false})\}$. The oracle outputs c.

   Give random oracle access to $\mathcal{A}$. Note that, when the verification is successful, we can derive that $y = g_1^{\alpha\beta}$.
5. The random oracle H_3 is programmed as follows: Define a list $\mathcal{L}_{H_3} = \emptyset$. For a query on y, if there exists a tuple (y, c), then output c. Otherwise choose a random $c \xleftarrow{R} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_3} = \mathcal{L}_{H_3} \cup (y, c)$. The oracle outputs c. Give random oracle access to $\mathcal{A}$.
6. Run the simulator of the secure DKG protocol to interact with the corrupted servers. Let QUAL be the non-disqualified servers determined after the generating phase. Assume the combined polynomial after the generating phase is f. In the extracting phase, for the ℓ-th polynomial, $\mathcal{B}$ constructs $f_\ell^\star$ to replace f_ℓ: $f_\ell^\star$ has the values $\alpha - \sum_{j \in \mathrm{QUAL} \setminus \{\ell\}} f_j(0)$, $f_\ell(1)$, $\ldots$, $f_\ell(t)$. $A_{i,j}, A_{\ell,j}^\star$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $0 \le j \le t$ can be computed similar to [GJKR07]. $B_{i,0}^\star$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $B_{\ell,0}^\star$ are computed as:
   - For $i \in \mathrm{QUAL} \setminus \{\ell\}$, $B_{i,0} = g_2^{a_{i,0}}$ since $\mathcal{B}$ has the coefficients of all the polynomials except the ℓ-th polynomial.
   - $B_{\ell,0}^\star = g_2^\alpha \prod_{i \in \mathrm{QUAL} \setminus \{\ell\}} B_{i,0}^{-1}$.

   The global public key is $\mathsf{pk} = B_{\ell,0}^\star \prod_{i \in \mathrm{QUAL} \setminus \{\ell\}} B_{i,0} = g_2^\alpha$. Let the final combined polynomial be $f^\star$. We know that $f^\star(0) = \alpha$, $f^\star(1) = f(1), \ldots, f^\star(t) = f(t)$. The corrupted servers that are included in QUAL have the shares $\mathsf{sk}_i = f^\star(i)$ and $\mathsf{vk}_i = g_1^{f^\star(i)}$ for $i \in \mathrm{QUAL} \cap C$. The shares of the honest servers are set to be $\mathsf{sk}_i = f^\star(i)$ and $\mathsf{vk}_i = g_1^{f^\star(i)}$ for $m+1 \le i \le \ell$. Note that, for the honest servers $t+1 \le i \le \ell$, the simulator cannot compute sk_i because it does not know the value of α, but can compute $\mathsf{vk}_i = g_1^{\alpha \lambda_{i,0,T}} \prod_{j \in T, j \ne 0} g_1^{f^\star(j)\lambda_{i,j,T}}$ with $T = \{0, 1, \ldots, t\}$.
7. Choose q_E t-degree random polynomials, $f^1, f^2, \ldots, f^{q_E - 1}$ such that for $1 \le j \le q_E - 1$, $f^j(0) = \alpha$ and for all $i \in C$ $f^j(i) = f^\star(i)$. Note that, we cannot compute $f^j(i)$ for $i \ge m+1$ because α is unknown, but we can compute $g_1^{f^j(i)}$ in the way similar to vk_i.
8. On an evaluation query (Eval, x, i) for an honest i, invoke random oracle H_1 to get $(x, r, h) \in \mathcal{L}_{H_1}$ and
   (a) return $(g_1^{f^j(i) \cdot r}, \pi_i)$ if $r \ne \perp$, where π_i is a simulated proof generated by calling the random oracle H_3
   (b) if $r = \perp$, choose $z_i \xleftarrow{R} \mathbb{G}_1$ and generate a simulated proof π_i using random oracle H_3 and return (z_i, π_i).

9. On the challenge query $(\mathsf{Challenge}, x^\star, \{(i, z_i^\star, \pi_i)\}_{i\in U}, V)$, where $|V| > t$ and $V \subseteq \mathrm{QUAL}$ and $U \subseteq V \cap C$ and $z_i^\star$ a set of evaluation shares from the corrupted servers, is answered as follows:
   (a) if $x^\star$ was not the $\eta^\star$-the query to H_1, then $\mathcal{B}$ aborts.
   (b) Otherwise do as follows:
      i. Run $\mathsf{VerifyEq}_{H_3}$ to check the proofs π_i for $i \in U$. If any check fails, output 0 and stop.
      ii. If there exists a tuple (y, c, true) in H_2, then set $v^\star = c$. Otherwise choose $v^\star \xleftarrow{R} \mathbb{Z}_q$ and update $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup \{\perp, v^\star, \mathsf{true}\}$. Depending on b do as follows:
         A. If $b = 0$ then return $v^\star$;
         B. Else return a uniform random.
10. Continue answering evaluation queries as before, but if $\mathcal{A}$ makes queries of the form $(\mathsf{Eval}, x^\star, i)$ for some $i \in \mathrm{QUAL} \setminus C$ and i is the $(t + 1 - m)$-th honest server that $\mathcal{A}$ contacted then output 0 and stop.
11. Receive a guess b' from $\mathcal{A}$.
12. If there exists a tuple (y, c, true) with $y \neq \perp$ in H_2, $\mathcal{B}$ outputs y as a solution to the co-CDH problem.

The probability that $\mathcal{B}$ does not abort is $1/q_{H_1}$ since $\eta^\star$ is uniformly and randomly chosen. Let's consider the case when `abort` does not happen. In this case, the challenge plaintext $x^\star$ is also the $\eta^\star$-th distinct query to H_1 and $H_1(x^\star) = g_1^\beta$. In Step 8b, to answer the evaluation queries on $x^\star$, we choose a random z_i. The adversary is allowed to make at most $t - m$ evaluation queries on $x^\star$. Let these evaluation queries be $(\mathsf{Eval}, x^\star, i_1), \ldots, (\mathsf{Eval}, x^\star, i_{t-m})$ and the returned randoms be $z_{i_1}, \ldots, z_{i_{t-m}}$. These values $(i_1, z_{i_1}), \ldots, (i_{t-m}, z_{i_{t-m}})$ together with $(0, \alpha\beta)$ and $(1, f^\star(1)\beta), \ldots,$ $(m, f^\star(m)\beta)$ implicitly defines an unique random polynomial $\hat{f}$ with $\hat{f}(0) = \alpha$, $\hat{f}(1) = f^\star(1), \ldots,$ $\hat{f}(m) = f^\star(m)$ and $\hat{f}$ is only used for computing evaluations on $x^\star$. Note that, it does not matter when the adversary makes less than $t-m$ evaluation queries on $x^\star$ since the polynomial $\hat{f}$ will never be explicitly computed. Therefore, $\mathcal{B}$ simulates $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(b)$ perfectly when aborts does not occur.

The rest of the analysis is similar to the one in the case $m = t$. We define an event C as when the adversary queries $y^\star$ to H_2 such that $e(y^\star, g_2) = e(g_1^\beta, g_2^\alpha)$, i.e., $y^\star = g_1^{\alpha\beta}$. We can prove that $\mathcal{A}$'s advantage $\mathbf{Adv}$ of distinguishing $b = 0$ and $b = 1$ is not bigger than $\Pr[C]$, i.e., $\mathbf{Adv} \leq \Pr[C]$. Since we assume $\mathcal{A}$ distinguishes $b = 0$ and $b = 1$ with non-negligible advantage $\mathbf{Adv}$, C happens with non-negligible probability $\Pr[C]$. Therefore $\mathcal{B}$ outputs y as a solution to the co-CDH problem with non-negligible probability $\Pr[C]/q_{H_1}$.