Non-Interactive Fully Distributed Verifiable Random Functions and their Applications

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

[1] Fetch.AI
[2] University of Birmingham

Abstract. We provide a systematic analysis of (Non-Interactive) Fully Distributed Verifiable Random Functions (DVRFs), including their syntax, definition of their privacy and security properties, and concrete constructions. We review how to build a decentralized random beacon from DVRFS, which is a key component for leader(s) election in distributed ledger technologies. We provide implementations and experimental evaluations of our concrete constructions using different cryptographic libraries.

Keywords Cryptography, Blockchain, Random Beacon, Distributed Computation

1 Introduction

In recent years there has been prolific development in the space of blockchain technologies. The blockchain, as a technology and data structure, can be traced back to the 1970s, but it had a strong re-emergence in 2008, when Bitcoin was introduced [Nak08]. Since then, a whole plethora of platforms relying on blockchains have seen the light of day. The various platforms often differ in their design choices, and thus with regards to which consensus protocol they rely on.

Several consensus protocols rely on associating the creation of blocks with a block producer. In many of the protocols the selection/election of a block producer requires a verifiable source of randomness. In order to avoid relying on a trusted party, a common approach is to provide for a mechanism that allows the distributed computation of a unpredictable and unbiased source of randomness.

1.1 Related Work

Next we review some of the main notions and primitives used in the literature to produce randomness in a multiparty setting.

Coin tossing One of the earliest approaches to compute a random value distributedly is known as *coin tossing*. Sometimes also known as coin flipping, it deals with the particular care of two distrustful parties, Alice and Bob, willing to create a shared random value. it is used in a variety of situations such as secret key exchange or mental poker [?]. In coin tossing As pointed out in [Blu83] a coin tossing protocol must rely on four key properties:

- Provided both sides follow the protocol there is exactly a 50-50 chance of getting heads
- Any observed attempt of cheating is provable (all messages need to be signed)
- When Bob flips coins she knows the outcome, whereas Bob doesn't
- After the tossings Alice should be able to prove the outcomes to Bob

Verifiable Random Functions This primitive is the public-key version of a keyed cryptographic hash $F_{\mathsf{sk}}(\cdot)$, where a trusted party evaluates $F_{\mathsf{sk}}(x)$ on a given input x in such a way that the output can be verified for correctness by any third party via an auxiliary proof Π_x. The secret key sk allows one to i) evaluate $F_{\mathsf{sk}}(\cdot)$ on an input x and ii) compute and provide proof of correctness Π_x of the evaluation $F_{\mathsf{sk}}(x)$. The proof correctness can be verified by means of running another algorithm, Verify. This algorithm takes as input the public key pk corresponding to sk, x, $F_{\mathsf{sk}}(x)$ and Π_x, and outputs accept or reject.

Secure Signature Schemes with Unique Signatures There are similarities between a VRF and an unforgeable signature scheme. On inputs a string x and a secret signing key SK_S, a signing algorithm outputs a signature $SIG(x)$, which is unpredictable given the signature scheme public key PK_S. However, despite its unpredictability, the signature $SIG(x)$ on string x is *publicly verifiable*. There are two issues that may prevent signature schemes from being used as VRF's:

1. signatures $SIG(x)$ may not be unique given x and PK_S
2. $SIG(x)$ is unpredictable but *not* pseudorandom (e.g. signatures could contain some bias and be distinguishable from a random distribution)

Most secure signature schemes are either probabilistic or history dependent (stateful). Both properties are incompatible with the uniqueness requirement.

It is hence necessary that a verification 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* [FZ12].

Exsiting random beacons SCRAPE [CD17] implements a publicly verifiable random beacon based on distributed secret sharing scheme. The protocol runs a fresh distributed secret sharing to generate a new verifiable random each time. However, the distributed secret sharing protocol is time-consuming and requires agreement. Therefore it is not optimal to generate random in this fashion. In addition, the security of the SCRAPE protocol as a whole is not formalised.

Our contribution...

Outline...

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 $g_1, g_2 \in \mathbb{G}$, the NIZK proof $(\mathsf{PrEq}_H, \mathsf{VerifyEq}_H)$ to show $\log_g x = \log_h y$ for $x, y \in \mathbb{G}$ is described as below [CP93]:

– $\mathsf{PrEq}_H(g, h, x, y, k)$ chooses $r \overset{R}{\leftarrow} \mathbb{Z}_q$, 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 $ch \overset{?}{=} H(g, h, x, y, com_1, com_2)$.

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 [MVR99]:

$\mathsf{KeyGen}(1^\lambda)$ 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 $X \leftarrow (\mathbb{G}, q, g, g^a, g^b)$ where $\langle \mathbb{G}, q, g \rangle \leftarrow \mathcal{I}(1^\lambda)$ and $a, b \overset{R}{\leftarrow} \mathbb{Z}_q^*$. We say that $\mathcal{I}$ satisfies the* Decisional Diffie-Hellman assumption *if for $r \overset{R}{\leftarrow} \mathbb{Z}_q^*$ 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 croups 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 $a, b \overset{R}{\leftarrow} \mathbb{Z}_q^*$ and $g_1 \overset{R}{\leftarrow} \mathbb{G}_1, g_2 \overset{R}{\leftarrow} \mathbb{G}_2$. 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* $\langle \mathsf{e}(\cdot,\cdot), \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q\rangle \leftarrow \mathcal{IG}(1^\lambda)$ *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. A message x is admissible if $x \in \mathcal{D}$. 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 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:*

$\mathsf{DistKG}(1^k, t, \ell)$ *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* $S_i \in \mathrm{QUAL}$*, secret key* sk_i *and verification key* vk_i*, a message* x*, and outputs either a triple* $s_x^i = (i, F_{\mathsf{sk}_i}(x), \pi_i)$*, where* $F_{\mathsf{sk}_i}(x)$ *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 message* 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 message* 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 message 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 $\mathcal{D}$; *provability*, meaning that the uniquely recovered value $v = F_{\mathsf{sk}}(x)$ passes the verification test; and *uniqueness*, meaning that for every message 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* $0 \le t < \ell$*, for every admissible message* x*, and any two* $(t+1)$*-subsets of partial evaluation shares* $\mathcal{V} \neq \mathcal{V}'$ *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* $0 \le t < \ell$*, for every admissible message* x*, and any* $(t+1)$*-subset* $\mathcal{E}$ *of partial evaluation shares such that* $\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, x, \mathcal{E}) = (v, \pi) \neq \mathtt{reject}$ *it holds that* $v \in \mathcal{R}$*.*

Provability: *for any integers* $0 \le t < \ell$*, every admissible message* x*, and any* $t+1$*-subset* $\mathcal{E}$ *of partial evaluation shares such that* $\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, x, \mathcal{E}) = (v_x, \Pi_x) \neq \mathtt{reject}$ *it holds that* $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, x, v_x, \Pi_x) = \mathtt{accept}$*.*

Uniqueness: *no integers* t, ℓ *with* $0 \le t < \ell$*, no admissible message* x*, no proofs* π, π' *and no values* $v, v' \in \mathcal{R}$ *can be found such that* $v \neq v'$ *and* $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, x, v, \pi) = \mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, x, v', \pi') = \mathtt{accept}$*.*

3.2 Security Properties: Correctness and Pseudorandomness

We give rigorous definitions of *correctness* and *pseudo-randomness* properties. Roughly speaking, correctness ensures the availability of computing the random function value on any admissible message 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* $0 \le t < \ell$*, a correctness adversary* $\mathcal{A}$ *chooses a collection* C *of servers to be corrupted with* $|C| \le t$*. 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 *which contains all the honest servers and some of the corrupted servers. Every (honest) server* $S_j \in S \setminus C$ *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* $S_i \in S \setminus C$ *and admissible message* x*, 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* $U \subseteq \mathrm{QUAL}$*, an admissible message* $x^\star$ *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.*

For pseudorandomness, we follow the latest general formal definition given in [AMMR18]. This 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 message (up to the threshold), and participate in computing the pseudorandom function on the challenge message.

Definition 10 (Pseudorandomness [AMMR18]). *A* DVRF *protocol* $\mathcal{V} = (\mathsf{DistKG}, \mathsf{PartialEval}, \mathsf{Combine}, \mathsf{Verify})$ *is* 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* $0 \le t < \ell$*, a correctness adversary* $\mathcal{A}$ *chooses a collection* C *of servers to be corrupted with* $|C| \le t$*. 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 *which contains all the honest servers and some of the corrupted servers. Every (honest) server* $S_j \in S \setminus C$ *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* $S_i \in S \setminus C$ *and admissible message* x*, 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* $U \subseteq$ QUAL *and* $|U| \ge t+1$*, and an admissible message* $x^\star$*, such that* $(\mathsf{Eval}, x^\star, i)$ *has been queried at most* $t - |C|$ *times for different* $S_i \in S \setminus C$*. 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* $\mathcal{D}$*.*

[Post-Challenge Evaluation] *In response to* $\mathcal{A}$*'s evaluation query* (Eval, x, i) *with* $x \ne x^\star$ *for some server* $S_i \in S \setminus C$ *and admissible message* x*, 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 message in the above experiment, we refer to it as *weak pseudorandomness*. This weaker notion of pseudorandomness has been informally or formally discussed in several literature [DY05,BLMR13].

4 DDH-based DVRF with non-compact proofs

Few VRF works detail how to generate the public and secret keys in a fully distributed manner with arbitrary threshold parameters (t, ℓ) such that $0 \le t \le \ell$. Previously, a fully distributed interactive DVRFs were studied in [KM13]. In this section we give a full descriptions and proofs of a fully distributed non-interactive verifiable random functions based on the Decisional Diffie-Hellman problem. The construction is obtained by using Pedersen's secure DKG protocol [Ped91] in the setup phase and the DDH-based VRF presented in [MBB+15,GRPV18]. For the security reductions, we use ideas from [GJKR07,AMMR18].

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 a DVRF for a pseudorandom function is defined as follows:

$\mathsf{DistKG}(1^\lambda, t, \ell)$ is run by ℓ participating severs $S = \{S_1, \ldots, S_\ell\}$. The full description of the protocol can be found in Figure 1 in Appendix A. 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 QUAL $\subseteq S$, a secret key $\mathsf{sk}_i = \sum_{j \in \mathrm{QUAL}} f_j(i)$ and a verification key $\mathsf{vk}_i = g^{\mathsf{sk}_i}$ for each $i \in$ QUAL, an implicit distributed secret value $\mathsf{sk} = \sum_{i \in \mathrm{QUAL}} a_{i,0}$, and a global public key $\mathsf{pk} = \prod_{i \in \mathrm{QUAL}} g^{a_{i,0}}$.

$\mathsf{PartialEval}(\mathsf{sk}_i, \mathsf{vk}_i, x)$ outputs $s_x^i = (i, v_i, \pi_i)$ for a message $x \in \{0,1\}^{a(\lambda)}$, 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 $r \overset{R}{\leftarrow} \mathbb{Z}_q$.

$\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, x, \mathcal{E})$ 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 for every $i \in I$ the following $\mathsf{VerifyEq}_{H_2}(g, \mathsf{vk}_i, H_1(x), v_i, \pi_i) = \texttt{accept}$ holds, where $(i, v_i, \pi_i) \leftarrow s_x^i$. If no such subset exists, outputs $\texttt{reject}$.
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 $\texttt{reject}$
3. Checks if $v = \prod_{i \in I} v_i^{\lambda_{0,i,I}}$; if so outputs $\texttt{accept}$; otherwise outputs $\texttt{reject}$.

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 obivious 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 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].

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

$\mathsf{Hyb}^{sim}_{\mathcal{A}}(b)$ is defined exactly the same as $\mathsf{Hyb}^{zk}_{\mathcal{A}}(b)$ 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. 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 $i \in \text{QUAL} \cap C$ 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: $\alpha - \sum_{j \in \text{QUAL}\setminus\{\ell\}} f_j(0)$, $f_\ell(1)$, $\ldots$, $f_\ell(m)$, $f_\ell(m+1)$, $\ldots$, $f_\ell(t)$. 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 $i \in \text{QUAL} \setminus \{\ell\}$ and $0 \leq j \leq t$ as below:

- For $i \in \text{QUAL} \setminus \{\ell\}$, the simulator has all the coefficients $a_{i,j}$ for $0 \leq j \leq t$ 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 $A^\star_{\ell,0} = g^{\alpha_0} \prod_{i \in \text{QUAL}\setminus\{\ell\}} A^{-1}_{i,0}$ 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 $\mathsf{pk} = A^\star_{\ell,0} \prod_{i \in \text{QUAL}\setminus\{\ell\}} A_{i,0} = g^\alpha$. $\mathsf{Hyb}^{sim}_{\mathcal{A}}(b)$ 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 $0 \leq k \leq q_E$ is defined exactly the same as $\mathsf{Hyb}^{sim}_{\mathcal{A}}(b)$ except the first k distinct 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 $1 \leq i \leq k$.

The indistinguishability between $\mathsf{Hyb}^{k-1}_{\mathcal{A}}(b)$ and $\mathsf{Hyb}^{k}_{\mathcal{A}}(b)$ for $1 \leq k \leq q_E$ 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 $y_0 \overset{R}{\leftarrow} \mathbb{G}, y_1 \overset{R}{\leftarrow} \mathbb{G}, \ldots, y_w \overset{R}{\leftarrow} \mathbb{G}$. 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: $\alpha_0 - \sum_{j \in \text{QUAL}\setminus\{\ell\}} f_j(0)$, $f_\ell(1)$, $\ldots$, $f_\ell(m)$, $\alpha_{m+1} - \sum_{j \in \text{QUAL}\setminus\{\ell\}} f_j(m+1)$, $\ldots$, $\alpha_t - \sum_{j \in \text{QUAL}\setminus\{\ell\}} f_j(t)$ 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 $i \in \text{QUAL} \setminus \{\ell\}$ and $0 \leq j \leq t$:

- For $i \in \text{QUAL}\setminus\{\ell\}$, $\mathcal{B}$ has all the coefficients $a_{i,j}$ for $0 \leq j \leq t$ for polynomial f_i, thus $A_{i,j} = g^{a_{i,j}}$ can be easily computed.

[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 $\ell_{\mathsf{hon}} > t \geq \ell_{\mathsf{corr}}$ where ℓ_{hon} is the number of the honest parties and ℓ_{corr} is the number of the corrupted servers.

– For the ℓ-th polynomial $f^\star_\ell$, $\mathcal{B}$ sets $A^\star_{\ell,0} = g^{\alpha_0} \prod_{i\in \text{QUAL}\setminus\{\ell\}} A^{-1}_{i,0}$ and $A^\star_{\ell,j} = (A^\star_{\ell,0})^{\delta_{0,j}} \prod_{i=1}^{m} g^{f_\ell(i)\cdot\delta_{i,j}}$ $\prod_{i=m+1}^{t}(g^{\alpha_i} \prod_{k\in \text{QUAL}\setminus\{\ell\}} g^{-f_k(i)})^{\delta_{i,j}}$ 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 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 $\mathsf{pk}_0 = A^\star_{\ell,0} \prod_{i\in \text{QUAL}\setminus\{\ell\}} A_{i,0}$ $= g^{\alpha_0}$ 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 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 $(m+1 \le i \le \ell)$, 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 $(1 \le i \le k-1)$ 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 $m+1 \le j \le t$, while answers with $(\prod_{i\in T} y_i^{\lambda_{j,i,T}}, \pi_j)$ for $j \ge t$, 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 polynomial), 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 $\mathsf{Hyb}^{k-1}_{\mathcal{A}}(b)$ perfectly; when $y_i \xleftarrow{R} \mathbb{G}$, the partial evaluation/challenge queries on x_k are answered using random polynomials, thus it simulates $\mathsf{Hyb}^{k}_{\mathcal{A}}(b)$. Therefore, $\mathsf{Hyb}^{k-1}_{\mathcal{A}}(b)$ and $\mathsf{Hyb}^{k}_{\mathcal{A}}(b)$ are indistinguishable under DDH assumption.

Finally $\mathsf{Hyb}^{q_E}_{\mathcal{A}}(0)$ and $\mathsf{Hyb}^{q_E}_{\mathcal{A}}(1)$ 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, but it cannot provide compact proofs π for verifying the pseudorandom value v, i.e., π is linear of the threshold t. Constructing DVRF based on cryptographic pairings can achieve compact proofs but would sacrifice the efficiency due to the time consuming operations (e.g., exponentiations, pairings) on pairing groups, e.g., the Dfinity DVRF [HMW18]. In this section, we shall first formalize Dfinity's pairing-based non-interactive DVRF which is informally described in [HMW18]. We notice that no formal treatment nor security proofs have been given to our knowledge. For Dfinity DVRF, we show that it achieves the weak pseudorandomness. Next, we shall propose a new pairing-based non-interactive DVRF, called $\mathsf{cp\text{-}DVRF}$, that can achieve pseudorandomness and provide compact proofs while its efficiency is comparable to the DDH-based DVRF.

5.1 Formalisation of the Dfinity DVRF

As mentioned in [HMW18], the Dfinity DVRF uses the biased DKG protocol where the adversary can bias the distribution of public keys [GJKR07]. In order to formalise the security of Dfinity DVRF, we replace the biased DKG protocol with the secure DKG protocol proposed in [GJKR07] and we shall prove that this construction satisfies the weak 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 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. The Dfinity DVRF $\mathcal{V}^{\mathsf{dfinity\text{-}DVRF}}$ 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 the verification keys $\mathsf{vk}_i = g_2^{\mathsf{sk}_i}$ and the global public key $\mathsf{pk} = \sum_{i \in \mathrm{QUAL}} g_2^{a_{i,0}}$ are generated in $\mathbb{G}_2$ using the generator $g_2 \in \mathbb{G}_2$. This means g, h are replaced with g_2, h_2 that are chosen from $\mathbb{G}_2$.

$\mathsf{PartialEval}(\mathsf{sk}_i, x)$ 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 $\mathcal{E} = \{s_x^{j_1}, \dots, s_x^{j_{|\mathcal{E}|}}\}$ of $|\mathcal{E}| \geq t+1$ partial evaluation candidates originating from $|\mathcal{E}|$ different servers, and obtains verification keys $\mathsf{vk}_{j_1}, \dots, \mathsf{vk}_{j_{|\mathcal{E}|}}$. Next,

1. Identifies an index subset $I = \{i_1, \dots, i_{t+1}\}$ 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 message 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 the same as 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 as 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 to 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 to the hypothesis.

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

Proof (Sketch). The full proof is given in Appendix C. Recall that weak pseudorandomness does not allow the adversary to query partial evaluations on the challenge message $x^\star$. Therefore, the weak pseudorandomness can be derived from the unforgeability of the threshold BLS signature. The definition of unforgeability of the threshold BLS does not allow an adversary to obtain any partial signature on the challenge message even if the adversary compromises less than t servers [Bol02]. Similar as 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 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 $g_1^{\alpha\beta}$. 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 $A_{i,j}, A^\star_{\ell,j}$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $0 \leq j \leq t$ 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 $\mathsf{pk} = g_2^\alpha$. Similarly, the simulator can compute $(\mathsf{sk}_i, \mathsf{vk}_i)$ for honest servers $m+1 \leq i \leq t$, while can 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 message, 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 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., $y^\star = g_1^{\alpha\beta}$. Apparently, 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 $\Pr[C]/q_{H_1}$.

5.2 Our proposed pairing-based non-interactive DVRF: cp-DVRF

The pseudorandomness does not follow from the unforgeability of threshold BLS signature because the pseudorandomness allows the adversary to query partial evaluations on the challenging message $x^\star$ while this is disabled in the unforgeability of threshold BLS. In order to prove the pseudorandomness, we modify the Dfinity DVRF as follows:

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

These changes help ease the security proof of the pseudorandomness. In the security reduction, when answering the partial evaluation queries on the challenging message, the responses may be randomised. If the verification key vk_i is in G_2, then the adversary can verify if an answer v_i is correct or not by checking the pairing $e(v_i, g_2) = e(H_1(x^\star), \mathsf{vk}_i)$. In comparison, when vk_i are in G_1, the NIZK used to validate v_i can be simulated, thus v_i can be randomly generated.

In our proposed scheme, the partial evaluations are validated using the NIZKs, but the pseudorandom value is still validated using pairing equations which provides a compact proof for the pseudorandom value. 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 makes our scheme $2.5 \sim 4.5$ faster than the Dfinity DVRF (see Section 6.2 for details). One may argue that the Dfinity DVRF allows the Combine algorithm to verify partial evaluations in batch which can reduce $2 \cdot |\mathcal{E}|$ pairings into 2 pairings. However, the batch verification is not suitable for the Combine algorithm: 1) batch verification only works when the verification is successful. When the verification fails, the partial evaluations still need to be verified one-by-one; 2) 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]; 3) 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.

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 $\mathcal{V}^{\mathsf{cp\text{-}DVRF}} = (\mathsf{DistKG}, \mathsf{PartialEval}, \mathsf{Combine}, \mathsf{Verify})$ 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 which are from $\mathbb{G}_1$.
- In Step 4, each server S_i with $i \in$ QUAL exposes $B_{i,0} = g_2^{a_{i,0}}$ via Feldman-VSS.
  - In Step 4(a), each server S_i with $i \in$ 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 $i \in$ 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 $\mathsf{pk} = \prod_{i \in \mathrm{QUAL}} B_{i,0}$.

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.

$\mathsf{PartialEval}(\mathsf{sk}_i, x)$ 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 $r \overset{R}{\leftarrow} \mathbb{Z}_q$.

$\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, x, \mathcal{E})$ 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 message x, the relation holds: $e(\pi, g_2) = e(H_1(x), \mathsf{pk})$ and $v = H_2(\pi)$. Otherwise output `reject`.

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

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

Theorem 6. $\mathcal{V}^{\mathsf{cp\text{-}DVRF}}$ *is 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 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 message $x^\star$. 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 message $x^\star$. To prove the 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 message $x(\neq x^\star)$ 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 $\alpha - \sum_{j \in \mathrm{QUAL} \setminus \{\ell\}} f_j(0)$, $f_\ell(1)$, ..., $f_\ell(t)$. $A_{i,j}, A^\star_{\ell,j}$ 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.
   - $B_{\ell,0} = g_2^\alpha \prod_{i \in \mathrm{QUAL} \setminus \{\ell\}} B_{i,0}^{-1}$.

The global public key is $\mathsf{pk} = B_{\ell,0} \prod_{i \in \text{QUAL} \setminus \{\ell\}} B_{i,0} = g_2^{\alpha}$. $\mathcal{B}$ 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 $i \in C$ $f^j(i) = s_i$. 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 $g_1^{f^j(i)}$ is computed similar to vk_i even when $f^j(i)$ is unknown and π_i is a simulated proof. For each evaluation query (Eval, x, i) on the challenge message $x^\star$, $\mathcal{B}$ 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 $(1, f(1)\beta)$, ..., $(m, f(m)\beta)$ implicitly defines an unique random polynomial $\hat{f}$ such that $\hat{f}(0) = \alpha$, $\hat{f}(1) = f(1)$, ..., $\hat{f}(m) = f(m)$, 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 **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 **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 leader at random in Proof-of-Stake blockchains (e.g. Dfinity [HMW18], Ethereum 2.0 [But18], OmniLedger [KJG+18]), without the need for a coordinator. The DRB is a special 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

$$\sigma_r \leftarrow F_{\mathsf{sk}}\left(\sigma_{r-1} || r\right),$$

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

$$\mathsf{Verify}(\mathsf{pk}, x, \sigma_r, \pi_x) = \begin{cases} 1 \text{ if true} \\ 0 \text{ otherwise} \end{cases}$$

where $x = \sigma_{r-1}||r$.

Choosing the initial seed. The seed initial σ_0 to bootstrap the DRB should be free from manipulation from an attacker. Previous DRB constructions have failed to provide an explicit initial seed or a convincing justification for lack of adversarial protection against bias. We propose $\sigma_0 := \mathsf{pk}$, 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 from Gennaro *et al.*'s DKG protocol [GJKR07].

Strong bias resistance. Uniqueness provides the strongest form of bias resistance, as it stands even against any adversary independently from the number of corrupted nodes.

6.1 Theoretical performance

In Table 1, we give analytical measurements for the size of the output of the partial evaluation algorithm and the combine algorithm. 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.

Protocol	cp-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		cp-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 DistKG algorithm, we compare the number of computations that each server performs in the generating phase and the extracting phase.

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 [Fet19] **self reference should be removed for anonymous submission** are shown in Table 3, where the dfinity-DVRF and cp-DVRF protocols are implemented using mcl library [Mis19] and RELIC [AG14], and ddh-DVRF using Libsodium [BD19]. 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.

Protocol	Curve	Security Level	Security Strength (bits)	Prime Field Size	Assumption
cp-DVRF	BN256	Standard	110	256	co-CDH XDH
	BLS12-381	Standard	128	381	
	BN384	Standard	128	384	
ddh-DVRF	Ristretto255	Strong	128	255	DDH
dfinity-DVRF	BN256	Standard	110	256	co-CDH
	BLS12-381	Standard	128	381	
	BN384	Standard	128	384	

Table 3: Curves and their associated properties examined in benchmarks

Protocol	Curve	Crypto Library	Random Generation (ms)	Time Ratio
cp-DVRF	BN256	mcl	7.33	0.69
	BLS12-381	mcl	18.95	1.78
	BN384	mcl	21.12	1.98
	BN256	RELIC	31.51	2.96
ddh-DVRF	Ristretto255	Libsodium	10.65	1
dfinity-DVRF	BN256	mcl	19.02	1.79
	BLS12-381	mcl	55.28	5.19
	BN384	mcl	60.31	5.66
	BN256	RELIC	138.26	12.99

Protocol	Curve	Crypto Library	Random Generation (ms)	Time Ratio
cp-DVRF	BN256	mcl	28.50	0.66
	BLS12-381	mcl	71.68	1.65
	BN384	mcl	80.59	1.86
	BN256	RELIC	128.96	2.97
ddh-DVRF	Ristretto255	Libsodium	43.35	1
dfinity-DVRF	BN256	mcl	74.65	1.72
	BLS12-381	mcl	217.73	5.02
	BN384	mcl	227.42	5.25
	BN256	RELIC	546.95	12.62

Table 4: Time ratios per individual node for each protocol in table 3 to generate one round of entropy for $\ell = 50$ (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.

The results of the benchmarks are shown in Table 4. We observe that the ddh-DVRF protocol with Ristretto255 outperforms the dfinity-DVRF protocol with curves offering the same security level by approximately a factor of 5. In the case of BN256, which has the same prime field size 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 RELIC. Between the protocols with compact proofs, the cp-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 cp-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 [Fet19] **self reference should be removed for anonymous submission**. The reference implementation 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 the both the ECDSA and Diffie-Hellman public keys for each server is published on a secure bulletin board. In addition to the secure channels the DistKG requires a secure bulletin for broadcasting information to all participants, which is implemented using the reliable broadcast protocol described in [CKPS01].

Conclusion...

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.

[BD19] 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.

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

[Blu83] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *SIGACT News*, 15(1):23–27, January 1983.

[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. Cryptology ePrint Archive, Report 2017/216, 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.

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

[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.

[Fet19] Fetch.ai. Consensus. `https://github.com/fetchai/research-consensus/tree/refactoring`, 2019.

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

[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.

[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.

[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.

[MVR99] Silvio Micali, Salil Vadhan, and Michael Rabin. Verifiable random functions. FOCS '99, pages 120–130, 1999.

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

[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.

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

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 and pairing-based DVRF.

B Security proofs for DDH-based DVRF

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

– **Generating phase:**
1. Each server S_i performs a $\mathsf{Pedersen\text{-}VSS}$ 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 1.
   (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 1.
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 $i \in \mathrm{QUAL}$ exposes $A_{i,0} = g^{a_{i,0}}$ via **Feldman-VSS**
   (a) Each server S_i with $i \in \mathrm{QUAL}$, 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 $i \in \mathrm{QUAL}$, 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 1 but not Equation 2.
   (c) For server S_i who receives at least one valid complaint, i.e., values which satisfy Equation 1 but not Equation 2, 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 $j \in \mathrm{QUAL}$ 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].

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:

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

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 obivious 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 $S = \{S_1,\cdots,S_\ell\}$ 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 which contains all the honest servers and some corrupted servers, a key pair $(\mathsf{sk}_i, \mathsf{vk}_i)$ for every honest server $S_i\in S\setminus C$, 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 $r\overset{R}{\leftarrow}\mathbb{Z}_q$ 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 $c\overset{R}{\leftarrow}\mathbb{Z}_q$ 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 $S_i\in S\setminus C$, 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 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 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 $S = \{S_1, \cdots, S_\ell\}$ 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 $S_i \in S \setminus C$, 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 $r \overset{R}{\leftarrow} \mathbb{Z}_q$ 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 $c \overset{R}{\leftarrow} \mathbb{Z}_q$ 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 $S_i \in S \setminus C$, 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 $\mathsf{VerifyEq}_H$ 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 $i \in S \setminus C$ 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 - $\mathsf{Hyb}_{\mathcal{A}}^{zk}(b)$ Define a hybrid $\mathsf{Hyb}_{\mathcal{A}}^{zk}(b)$ 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. $\mathsf{Hyb}_{\mathcal{A}}^{zk}(b)$ is indistinguishable from $\mathsf{PseudoRand}_{\mathcal{A}}(b)$ for any PPT $\mathcal{A}$ and $b \in \{0, 1\}$ due to the zero-knoweldge property of NIZKs.

Second hybrid - $\mathsf{Hyb}_{\mathcal{A}}^{sim}(b)$ Define a hybrid $\mathsf{Hyb}_{\mathcal{A}}^{sim}(b)$ which is similar to $\mathsf{Hyb}_{\mathcal{A}}^{zk}(b)$ except that the running of securet 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 $f_i(x) = a_{i,0} + a_{i,1}x + \cdots + a_{i,t}x^t$ and $f'_i(x) = b_{i,0} + b_{i,1}x + \cdots + b_{i,t}x^t$ 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 $f_i(x), f'_i(x)$ 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 $0 \leq j \leq t$.
   – 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 $0 \leq j \leq t$ 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 $0 \leq j \leq t$.

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 $j \leq t$, the simulator can compute j's secret key $\mathsf{sk}_j = \sum_{i \in \text{QUAL}} s_{i,j}$ as well as the corresponding verification key vk_j. This is because the values of $s_{i,j}$ for $i \in \text{QUAL} \setminus C$ and $1 \leq j \leq t$ are essentially chosen by the simulator.

For each honest server $j \in \text{QUAL} \setminus C$ and $j > t$, the simulator is not able to computer j's secret key. This is because, the simulator chooses random polynomials for all the honest servers except the last honest server, i.e., ℓ-th server. For the ℓ-th server, the simulator cannot obtain its $f_\ell(0)$ since its value is determined by the equation $f_\ell(0) = \alpha - \sum_{i \in \text{QUAL} \setminus \{\ell\}} f_i(0)$ where α is unknown.This means the simulator cannot compute $s_{\ell,j} = f_\ell(j)$ for $j > t$. As a result, the simulator cannot compute the secret keys $\mathsf{sk}_j = \sum_{i \in \text{QUAL}} s_{i,j}$ for any $j > t$. However, the simulator can still compute the corresponding verification keys $\mathsf{vk}_j = g^{\sum_{i \in \text{QUAL}} s_{i,j}}$ for $j > t$ as follows: compute $Z = g^{\sum_{i \in \text{QUAL} \setminus \{\ell\}} s_{i,j}}$ and then $\mathsf{vk}_j = g^{f_\ell(j)} \cdot Z$. It is easy to see that the simulator has all the required $s_{i,j}$ to compute the value of Z, and $g^{f_\ell(j)} = \prod_{0 \leq i \leq t} g^{f_\ell(i) \cdot \lambda_{j,i,T}} = (g^\alpha)^{\lambda_{j,0,T}} (g^{-\sum_{i \in \text{QUAL} \setminus \{\ell\}} f_i(0)})^{\lambda_{j,0,T}} \prod_{1 \leq i \leq t} g^{f_\ell(i) \cdot \lambda_{j,i,T}}$ where $T = \{0, 1, \cdots, t\}$ and $\lambda_{j,i,T}$ are lagrange coefficents.

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 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 $1 \leq j \leq k$. 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 message in the evaluation/challenge queries, then return $\mathsf{pc}(x, j, i)$ defined as follows:

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

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

The challenge query $(\mathsf{Challenge}, x^\star, \{(i, z_i^\star, \pi_i)\}_{i \in U}, V)$ with $|V| > t$ 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:

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

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

Proof. We show that if there exists a PPT adversary $\mathcal{A}$ that can distinguish between the hybrids $\mathsf{Hyb}^{k-1}_{\mathcal{A}}(b)$ and $\mathsf{Hyb}^{k}_{\mathcal{A}}(b)$ with non-negligible probability then we can construct a PPT adversary $\mathcal{B}$ to break 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 $|C| \leq t$ 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 $c \overset{R}{\leftarrow} \mathbb{Z}_q$ 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. The generating phase is run exactly the same as described in Figure 1. The generating phase determines a set of non-disqualified servers QUAL. $\mathcal{B}$ acts on behalf of all the honest servers $\{m+1, \dots, \ell\}$, while the adversary controls all the corrupted servers $\{1, \dots, m\}$. The generating phase is run exactly the same as described in Figure 1. The generating phase determines a set of non-disqualified servers QUAL. 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 $f^\star_\ell(x)$, is constructed to take the values: $\alpha_0 - \sum_{j \in \text{QUAL} \setminus \{\ell\}} f_j(0)$, $f_\ell(1)$, $\dots$, $f_\ell(m)$, $\alpha_{m+1} - \sum_{j \in \text{QUAL} \setminus \{\ell\}} f_j(m+1)$, $\dots$, $\alpha_t - \sum_{j \in \text{QUAL} \setminus \{\ell\}} f_j(t)$. For $0 \le j \le t$, we show how to construct $A_{i,j}$ 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 $A^\star_{\ell,0} = g^{\alpha_0} \prod_{i \in \text{QUAL} \setminus \{\ell\}} A^{-1}_{i,0}$ and $A^\star_{\ell,j} = (A^\star_{\ell,0})^{\delta_{0,j}} \prod_{i=1}^{m} g^{f_\ell(i) \cdot \delta_{i,j}} \prod_{i=m+1}^{t} (g^{\alpha_i} \prod_{k \in \text{QUAL} \setminus \{\ell\}} g^{-f_k(i)})^{\delta_{i,j}}$ where $\delta_{i,j}$ is the coefficient of x^j in the lagrange basis polynomial $\lambda_{i,T}(x)$ with $T = \{0, 1, \dots, t\}$.

   Let the final combined polynomial be $f^\star$. We can see that $f^\star(0) = \alpha_0$, $f^\star(1) = f(1), \dots, f^\star(m) = f(m)$, $f^\star(m+1) = \alpha_{m+1}, \dots, f^\star(t) = \alpha_t$. 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 $\alpha_0, \alpha_{m+1}, \dots, \alpha_t$ are unknown. For the honest servers $t+1 \le i \le \ell$, we also cannot compute their shares sk_i because $\alpha_0, \alpha_{m+1}, \dots, \alpha_t$ are unknown, but we can compute their verification keys as $\mathsf{vk}_i = g^{\alpha_0 \lambda_{0,0,T}} \prod_{j \in T, j \ne 0} g^{f^\star(j) \lambda_{i,j,T}}$ with $T = \{0, \dots, t\}$. 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, $f^1, f^2, \dots, f^{k-1}$ such that for all $i \in C$ and $1 \le j \le k-1$, $f^j(i) = f(i)$.
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 $y_0, y_{m+1}, \dots, y_t$ 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 $T = \{0, 1, \dots, t\}$.
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, if x is the j-th distinct value, then return $\mathsf{pc}(x, j, i)$.
10. On the challenge query $(\mathsf{Challenge}, x^\star, \{(i, z^\star_i, \pi_i)\}_{i \in U}, V)$, where $|V| > t$ 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:
    (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 $(z^\star_i, \cdot) = \mathsf{pc}(x^\star, j^\star, i)$ 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 $i \in S \setminus C$ 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 $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(0)$ is indistinguishable from $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(1)$. 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 $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(0)$ and $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(1)$ are indistinguishable.

C Security proofs for Dfinity DVRF

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

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

Similar to Theorem 2, we construct $\mathsf{Hyb}^{sim}(b)$ to simulate the DKG protocol and the original weak 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 breaks the $\mathsf{Hyb}^{sim}(b)$, 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 $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, $\alpha, \beta \overset{R}{\leftarrow} \mathbb{Z}_q$. $\mathcal{B}$'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 $S = \{S_1, S_2, \ldots, S_\ell\}$ to $\mathcal{A}$.
2. $\mathcal{A}$ chooses a set C of servers with $|C| = t$ to corrupt and send C to $\mathcal{B}$. W.l.o.g., assume $C = \{1, 2, \ldots, t\}$.
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 $\eta^\star \overset{R}{\leftarrow} [q_{H_1}]$ 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 $r \overset{R}{\leftarrow} \mathbb{Z}_q$ 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 $c \overset{R}{\leftarrow} \mathbb{Z}_q$ 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 $c \overset{R}{\leftarrow} \mathbb{Z}_q$ 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. 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 $f_\ell^\star(x)$ to replace $f_\ell(x)$, the simulator sets $f_\ell^\star(x)$ 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 $\mathsf{Hyb}_{\mathcal{A}}^{sim}(b)$ in Theorem 2 except they are constructed on $\mathbb{G}_2$. The global public key is $\mathsf{pk} = g_2^\alpha$. 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 $V \subseteq \mathrm{QUAL}$ 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. 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 $v^\star \overset{R}{\leftarrow} \mathbb{Z}_q$ 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 message $x^\star$ 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 weak pseudorandomness. In other words, the Step 6b will never be executed. In this case $\mathcal{B}$ simulates the weak 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

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

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 our pairing-based DVRF: cp-DVRF

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

Proof. Similar to Theorem 2, we construct $\textsf{Hyb}^{zk}(b)$ to simulate all the NIZKs using random oracle H_3 and construct $\textsf{Hyb}^{sim}(b)$ to simulate the DKG protocol. Similarly, we have that the original pseudorandomness game and $\textsf{Hyb}^{zk}(b)$ are indistinguishable due to the zero-knowledge property of NIZKs, and $\textsf{Hyb}^{zk}(b)$ and $\textsf{Hyb}^{sim}(b)$ 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 message $x^\star$. Given an adversary $\mathcal{A}$ that distinguishes $\textsf{Hyb}^{sim}(0)$ and $\textsf{Hyb}^{sim}(1)$, 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 $c \overset{R}{\leftarrow} \mathbb{Z}_q$ 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 $f^\star_\ell$ to replace f_ℓ, the simulator sets up the points $\alpha - \sum_{j \in \text{QUAL} \setminus \{\ell\}} f_j(0)$, $f_\ell(1)$, $\ldots$, $f_\ell(t)$ for $f^\star_\ell(x)$. The $A_{i,j}, A^\star_{\ell,j}$ and $B^\star_{i,0}$ for $i \in \text{QUAL} \setminus \{\ell\}$ and $0 \leq j \leq t$ can be easily computed as $A_{i,j}, A^\star_{\ell,j}$ for $i \in \text{QUAL} \setminus \{\ell\}$ 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 $i \in \text{QUAL} \setminus \{\ell\}$ and $B^\star_{\ell,0}$ are computed as:
  - For $i \in \text{QUAL} \setminus \{\ell\}$, $B_{i,0} = g_2^{a_{i,0}}$ since $\mathcal{B}$ has the coefficients of all the polynomials except the ℓ-th polynomial.
  - $B_{\ell,0} = g_2^\alpha \prod_{i \in \text{QUAL} \setminus \{\ell\}} B^{-1}_{i,0}$.

  The global public key $\mathsf{pk} = \prod_{i \in \text{QUAL}} B_{i,0} = g_2^\alpha$. 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 $i \in \text{QUAL} \cap C$. 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 $T = \{0, 1, \ldots, t\}$.
- 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 $(\mathsf{Challenge}, x^\star, \{(i, z^\star_i, \pi_i)\}_{i \in U}, V)$, where $|V| > t$ 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 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 $m < t$: when $m < t$, the adversary is allowed to make up to $t - m (\geq 1)$ evaluation queries on the challenge message $x^\star$. To prove the 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 will show that

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

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

Lemma 2. *For $1 \le k \le q_E$, $\mathsf{Hyb}^{k-1}_{\mathcal{A}}(b)$ and $\mathsf{Hyb}^{k}_{\mathcal{A}}(b)$ 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 $\mathsf{Hyb}^{k-1}_{\mathcal{A}}(b)$ and $\mathsf{Hyb}^{k}_{\mathcal{A}}(b)$ with non-negligible probability then we can construct a PPT adversary $\mathcal{B}$ to break 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 $i \in \{1, \dots, w\}$ 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 $S = \{1, 2, \dots, \ell\}$ to $\mathcal{A}$.
2. $\mathcal{A}$ chooses a set $C = \{1, 2, \dots, m\}$ 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 $\eta^\star \overset{R}{\leftarrow} [q_{H_1}]$ 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 $r \overset{R}{\leftarrow} \mathbb{Z}_q$ 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 $c \overset{R}{\leftarrow} \mathbb{Z}_q$ 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 $c \overset{R}{\leftarrow} \mathbb{Z}_q$ 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 $\gamma \overset{R}{\leftarrow} \mathbb{Z}_q$. In the extracting phase, the ℓ-th polynomial $f_\ell^\star$ is constructed using the points $\gamma - \sum_{j\in \mathrm{QUAL}\setminus\{\ell\}} f_j(0)$, $f_\ell(1)$, $\dots$, $f_\ell(m)$, $\alpha_{m+1} - \sum_{j\in \mathrm{QUAL}\setminus\{\ell\}} f_j(m+1)$, $\dots$, $\alpha_t - \sum_{j\in \mathrm{QUAL}\setminus\{\ell\}} f_j(t)$ where $\alpha_{m+1}, \dots, \alpha_t$ are from the XDH problem. $A_{i,j}, A^\star_{\ell,j}$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $0 \le j \le t$ can be computed similar to Lemma 1. $B^\star_{i,0}$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $B^\star_{\ell,0}$ are computed as:

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

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 $\mathsf{vk}_i = g_1^{\gamma\lambda_{0,0,T}} \prod_{j \in T, j \neq 0} g_1^{f^\star(j)\lambda_{i,j,T}}$ 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 $1 \leq j \leq k-1$, $f^j(i) = f(i)$.
8. Compute $\bar{z}_0 = g_1^{\beta\gamma}$ and $\bar{z}_i = (g_1^\beta)^{f(i)}$ for $i \in C$. Set $\bar{z}_i = y_i$ for $m+1 \leq i \leq t$ where $y_{m+1}, \ldots, y_t$ are from the XDH problem. Then for all $m+1 \leq i \leq \ell$, 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 $j < k$, where $g_1^{f^j(i)}$ can be computed similar to vk_i even if $f^j(i)$ may be unknown.
   - 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 $V \subseteq \text{QUAL}$ 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 $\mathsf{VerifyEq}_{H_3}$ 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 $i \in S \setminus C$ 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 $\mathsf{Hyb}^{q_E}_{\mathcal{A}}(0)$ is indistinguishable from $\mathsf{Hyb}^{q_E}_{\mathcal{A}}(1)$. In $\mathsf{Hyb}^{q_E}_{\mathcal{A}}(b)$, 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. *$\mathsf{Hyb}^{q_E}_{\mathcal{A}}(0)$ and $\mathsf{Hyb}^{q_E}_{\mathcal{A}}(1)$ are indistinguishable under co-CDH assumption in the random oracle model.*

Proof. 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 $g_1^{\alpha\beta}$:

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 $S = \{S_1, S_2, \ldots, S_\ell\}$ 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 $\alpha - \sum_{j \in \mathrm{QUAL} \setminus \{\ell\}} f_j(0)$, $f_\ell(1)$, $\ldots$, $f_\ell(t)$. $A_{i,j}, A^\star_{\ell,j}$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $0 \le j \le t$ 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.

  – $B_{\ell,0} = g_2^{\alpha} \prod_{i \in \text{QUAL} \setminus \{\ell\}} B_{i,0}^{-1}$.

  The global public key $\mathsf{pk} = B_{\ell,0} \prod_{i \in \text{QUAL} \setminus \{\ell\}} B_{i,0} = g_2^{\alpha}$. 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 $i \in \text{QUAL} \cap C$. The shares of the honest servers are set to be $\mathsf{sk}_i = f(i)$ and $\mathsf{vk}_i = g_1^{f(i)}$ for $m+1 \leq i \leq \ell$. 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 vk_i as $g_1^{\alpha\lambda_{i,0,T}} \prod_{j \in T, j \neq 0} g_1^{f(j)\lambda_{i,j,T}}$ with $T = \{0, 1, \ldots, t\}$.

7. Choose q_E t-degree random polynomials, $f^1, f^2, \ldots, f^{q_E - 1}$ such that for $1 \leq j \leq q_E - 1$, $f^j(0) = \alpha$ and for all $i \in C$ $f^j(i) = f^{\star}(i)$.
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 $r \neq \perp$, where $g_1^{f^j(i)}$ can be computed similar to vk_i even if $f^j(i)$ may be unknown and 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 $V \subseteq \text{QUAL}$ 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 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 $i \in S \setminus C$ 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 message $x^{\star}$ 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 $(1, f(1)\beta)$, $\ldots$, $(m, f(m)\beta)$ implicitly defines an unique random polynomial $\hat{f}$ with $\hat{f}(0) = \alpha$, $\hat{f}(1) = f(1)$, $\ldots$, $\hat{f}(m) = f(m)$ 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 $\Pr[C]/q_{H_1}$.