Key-Homomorphic and Aggregate Verifiable Random Functions

Giulio Malavolta

Bocconi University

Abstract

A verifiable random function (VRF) allows one to compute a random-looking image, while at the same time providing a unique proof that the function was evaluated correctly. VRFs are a cornerstone of modern cryptography and, among other applications, are at the heart of recently proposed proof-of-stake consensus protocols. In this work we initiate the formal study of *aggregate VRFs*, i.e., VRFs that allow for the aggregation of proofs/images into a small digest, whose size is *independent* of the number of input proofs/images, yet it still enables sound verification. We formalize this notion along with its security properties and we propose two constructions: The first scheme is conceptually simple, concretely efficient, and uses (asymmetric) bilinear groups of prime order. Pseudorandomness holds in the random oracle model and aggregate pseudorandomness is proven in the algebraic group model. The second scheme is in the standard model and it is proven secure against the learning with errors (LWE) problem.

As a cryptographic building block of independent interest, we introduce the notion of *key homomorphic VRFs*, where the verification keys and the proofs are endowed with a group structure. We conclude by discussing several applications of key-homomorphic and aggregate VRFs, such as distributed VRFs and aggregate proof-of-stake protocols.

Contents

1 Introduction

A verifiable random function (VRF) is a keyed function that satisfies the following cryptographic properties:

- *Pseudorandomness:* The evaluation of the function at any point is computationally indistinguishable from uniform, provided that the distinguisher is not given the secret key.

- *Verifiability:* For any given image, there exists a unique proof of correct evaluation of the function that can be publicly verified, without revealing any additional information about the secret key.

VRFs were first introduced in the work of Micali, Rabin and Vadhan [MRV99], and have been subject of an intense research effort. Over the years, a large amount of VRF constructions have emerged [DY05, HJ16, Lys02, Jag15, HW10, GHKW17, Bit17, GLOW21], improving on the original proposal in terms of computational assumptions, and concrete efficiency. This progress was accompanied by a surge in applications built on top of VRFs, such as randomness generation [DY05, HJ16], e-cash [BCKL09, ASM07], key-escrow [JS04], zero-knowledge [Lis05], network security [GNP+15], and more recently proof-of-stake [GHM+17, DGKR18] and distributed consensus [HMW18] protocols. Given the large applicability of this primitive, it is safe to say that VRFs have a central role in modern cryptography. However, somewhat surprisingly, *aggregation properties* of VRFs have remained considerably understudied, and we lack even a good set of definitions for their precise security properties. This is in stark contrast with the situation for aggregate signatures [BGLS03], where aggregation algorithms and the practical benefits of this primitive have received a lot of attention in the recent years. The goal of this work is to fill this gap in our current understanding, and systematically study the aggregation properties of VRFs.

1.1 Our Contributions

In this work, we initiate the formal study of aggregate VRFs. Our contributions can be summarized as follows.

(a) Definitions. We formally define the notion of aggregate VRFs and propose new security definitions for uniqueness, binding, and pseudorandomness (Sec. 3.3), in the presence of maliciously generated keys. The precise notions turn out to be surprisingly subtle to identify.

(b) Pairing-Based Aggregate VRF. We present a new VRF construction based on asymmetric bilinear pairings (Sec. 4), which satisfies the strong notion of unique proofs (unconditionally), and pseudorandomness assuming the hardness of a variant of the bilinear Diffie-Hellman problem, in the random oracle model. Our scheme is conceptually simple and *concretely* efficient: All algorithms perform only a handful of group operations and the verification is dominated by the computation of three pairings. We propose an algorithm to aggregate VRF images and proofs (Sec. 4.2) and we show that it preserves pseudorandomness and (aggregate) uniqueness, in the algebraic group model [FKL18].

(c) Lattice-Based Aggregate VRF. We present an aggregate VRF construction using general-purpose cryptographic tools (Sec. 5), that we can prove secure assuming the hardness of the standard learning with errors (LWE) problem. Our proof is in the standard model and we view this construction as our main technical contribution.

(d) Key-Homomorphic VRF. Along the way, we introduce the notion of key-homomorphic VRF (Sec. 3.2), which may be of independent interest. A key-homomorphic VRF is a standard VRF, where the key space $\mathcal{K}$ is endowed with a group structure, with $\oplus$ denoting the group operation. Given a point x, two images $y_0 = \mathsf{Eval}(k_0, x)$ and $y_1 = \mathsf{Eval}(k_1, x)$, and two proofs π_0 and π_1, there exists an efficient procedure that computes

$$y = \mathsf{Eval}(k_0 \oplus k_1, x) \quad \text{and} \quad \pi = \mathsf{Prove}(k_0 \oplus k_1, x).$$

We show that our pairing-based VRF is key-homomorphic for linear function and we consider several extensions of this base construction, such as a threshold variant (Sec. 4.4) and an anonymous version (Sec. 4.3).

(e) Applications. To substantiate the usefulness of our newly defined primitives, we describe how key-homomorphic and aggregate VRFs yield protocols for several applications (Sec. 6). For instance, we show how key-homomorphic VRFs imply a simple distributed VRFs, key homomorphic pseudorandom functions [BLMR13] with an additional verification property, and aggregate proof-of-stake consensus protocols.

1.2 Technical Outline

We give a brief technical outline of the main ideas behind our work. We start by describing a construction of key-homomorphic VRF and how it naturally leads to aggregation properties. We then zoom in the security of aggregate VRFs, and subsequently propose an alternative construction with security against the LWE assumption. We conclude by sketching one possible application of aggregate and key-homomorphic VRFs.

Key-Homomorphic VRF. Our starting point is a simple VRF based on asymmetric bilinear pairings of prime order p, with generators (g_1, g_2) and a uniformly sampled group element $S = g_1^s$, that we assume to be available to all participants. We stress that, while technically our scheme requires a common reference string to store S, the setup is completely transparent, and can be sampled with public coins by hashing into the group. A secret-verification key pair for our VRF is defined to be

$$k \leftarrow\$ \mathbb{Z}_p \text{ and } \mathsf{vk} = S^k.$$

On the other hand, the evaluation at point x is computed by pairing:

$$y = e\left(g_1, \mathcal{H}(x)\right)^k$$

where $\mathcal{H}$ is a hash function (modeled as a random oracle) that maps bitstrings into $\mathbb{G}_2$ elements. To prove that the evaluation was done correctly, one can simply compute $\pi = \mathcal{H}(x)^k$, which can be efficiently verified to be a valid proof, by checking

$$y \overset{?}{=} e\left(g_1, \pi\right) \quad \text{and} \quad e\left(\mathsf{vk}, \mathcal{H}(x)\right) \overset{?}{=} e\left(S, \pi\right).$$

It can be easily verified that both the proof and the image are uniquely determined by the verification key, which implies that the construction satisfies uniqueness. We can also show that the VRF is pseudorandom, with an argument similar to [BLS04], with the crucial difference that we require a different variant of the bilinear Diffie-Hellman assumption. One interesting property of this scheme is that it is *key homomorphic* for linear functions. To see why, it suffices to observe that

$$\begin{aligned}\mathsf{Eval}(k_0, x) \cdot \mathsf{Eval}(k_1, x) &= e\left(g_1, \mathcal{H}(x)\right)^{k_0} \cdot e\left(g_1, \mathcal{H}(x)\right)^{k_1} \\ &= e\left(g_1, \mathcal{H}(x)\right)^{k_0+k_1} \\ &= \mathsf{Eval}\left(k_0 + k_1, x\right)\end{aligned}$$

and similarly

$$\begin{aligned}\mathsf{Prove}(k_0, x) \cdot \mathsf{Prove}(k_1, x) &= \mathcal{H}(x)^{k_0} \cdot \mathcal{H}(x)^{k_1} \\ &= \mathcal{H}(x)^{k_0+k_1} \\ &= \mathsf{Prove}\left(k_0 + k_1, x\right)\end{aligned}$$

and the claim follows by linearity.

Aggregation. The scheme described above suggests a natural aggregation algorithm for VRF images and proofs: Simply apply the group operation to the images and proofs. By the key-homomorphism of the construction, the aggregate is still perfectly verifiable. Unfortunately this proposal suffers from *adaptive attacks*, that is, an attacker could sample maliciously one of the keys included in the aggregate, in such a way that it cancels out the contribution of some honest key. To make things worse, this simple solution is not even *collision resistant*, since any attacker can find two different set of preimages that result in the same aggregate image. We view these two attacks as strong indication that a different solution is needed.

Before describing our actual solution, let us give a somewhat more detailed account of the properties that we consider in this work for aggregate VRFs. The first notion that we consider is that of (i) *aggregate pseudorandomness*, which guarantees that any image of the aggregate VRF is still pseudorandom, so long as at least one of the parties involved is honest. This property is useful for protocols where the aggregate VRF is used as a random beacon, and we want it to be unbiasable by dishonest parties. The second property that we consider is that of (ii) *aggregate binding*, which states that it must be computationally hard to find an aggregate proof for an invalid set of images. This property mimics the collision resistance of a regular hash function, and it is important for scenarios where proofs are aggregated, but we want to prevent the adversary from changing the corresponding VRF images after the fact. Finally, we enforce the notion of (iii) *aggregate uniqueness*, which states that for an aggregate image, there should still exist a *unique* aggregate proof. I.e., the result of the aggregate algorithm is still a VRF.

Give this premise, let us now describe our aggregation algorithm. Instead of taking a simple product, we aggregate proofs and images by computing

$$\tilde{y} = \prod_{i=1}^{n} y_i^{r_i} \quad \text{and} \quad \tilde{\pi} = \prod_{i=1}^{n} \pi_i^{r_i}$$

where $r_1, \ldots, r_n$ are random coefficients computed as a deterministic function of the input. We are then able to show that this aggregation algorithm satisfies both of the desired properties, in the algebraic group model. We specifically mention here that, while this approach is inspired by the work of [BDN18], our analysis is completely different, due to the fact that we are proving a decision property, rather than a search property.

Aggregate VRF from LWE. The construction as described above suffers from two drawbacks: It is proven secure in the random oracle model, and furthermore it is broken by quantum attacks. As a contribution of independent interest, we show a construction of an aggregate VRF (satisfying all of the properties outlined above) from general-purpose tools, that we can prove secure assuming the standard LWE assumption. Our scheme compiles any (non-aggregate) VRF and uses three main ingredients:

- *Function-Binding Hash:* A tree-based hash function Hash, where the root of the tree is statistically bound to any *node* (including intermediate ones) of the tree. The node is fixed at setup time, and the hash satisfies *mode indistinguishability*, in the sense that setups for different nodes are computationally indistinguishable. This notion is new to our work, and generalizes recent hash functions [HW15, FWW23, BBK$^+$23] beyond one-bit predicates. As a contribution of independent interest, we show how to build such a hash function using rate-1 FHE [GH19, BDGM19].

- *Somewhere Extractable Batch Arguments:* Somewhere extractable batch arguments (SE-BARG) for NP for batches of n-many NP statements. The somewhere extractability property says that there exists an index $i \in \{1, \ldots, n\}$ (fixed at setup time), such that, given a trapdoor, one can extract the witness for the i-th statement for the BARG. Furthermore, the index i is computationally hidden. Such SE-BARGs can be built from a variety of assumptions [CJJ21a, CJJ21b, WW22, KLVW23, CGJ$^+$23], including LWE.

- *Compute-and-Compare Obfuscation:* A compute and compare program, consists of a function f and a target y, and accepts an input x if $f(x) = y$. It is known how to obfuscate such functionalities, provided that y has enough min-entropy conditioned on f, assuming LWE [WZ17, GKW17].

Armed with these tools, we can proceed to outline our aggregation algorithms. To compress images $y_1, \ldots, y_n$ of a given (non-aggregate) VRF, we apply the function

$$\tilde{C} \circ \mathsf{Hash}(y_1, \ldots, y_n)$$

where $\tilde{C}$ is the obfuscation of a dummy (always rejecting) circuit. On the other hand, to aggregate proofs, we compute *two separate BARGs* (σ_0, σ_1) that certify that for each committed y_i, there exists a valid VRF proof π_i. Verification simply checks the validity of the two BARGs. Aggregate binding follows naturally from the collision resistance of Hash, and we discuss the proof of aggregate pseudorandomness and aggregate uniqueness in the following.

Aggregate Pseudorandomness via Randomness Extraction. We show that the construction is pseudorandom, as long as at least one of the input y_i is also pseudorandom. First, we switch the function Hash in binding mode for the i-th index. In the next hybrid, we switch y_i to be uniform and independently sampled. At this point, it is tempting to conclude that, since Hash is statistically bound to y_i, then its output is uniformly distributed. Unfortunately, this intuition is *flawed*, since the other inputs $y_{j \neq i}$ are chosen adversarially, and may bias the output of $\mathsf{Hash}(y_1, \ldots, y_n)$. In other words, we need to argue that Hash acts as a *randomness extractor* for seed-dependent sources. This is in general not possible, as shown by strong impossibility results on randomness extractors for sources that may depend on the random seed [DVW20].

This is where the obfuscation $\tilde{C}$ enters into the picture. To solve this conundrum, we proceed in two steps: First (i) we switch $\tilde{C}$ to be the obfuscation of the circuit that *extracts* y_i from

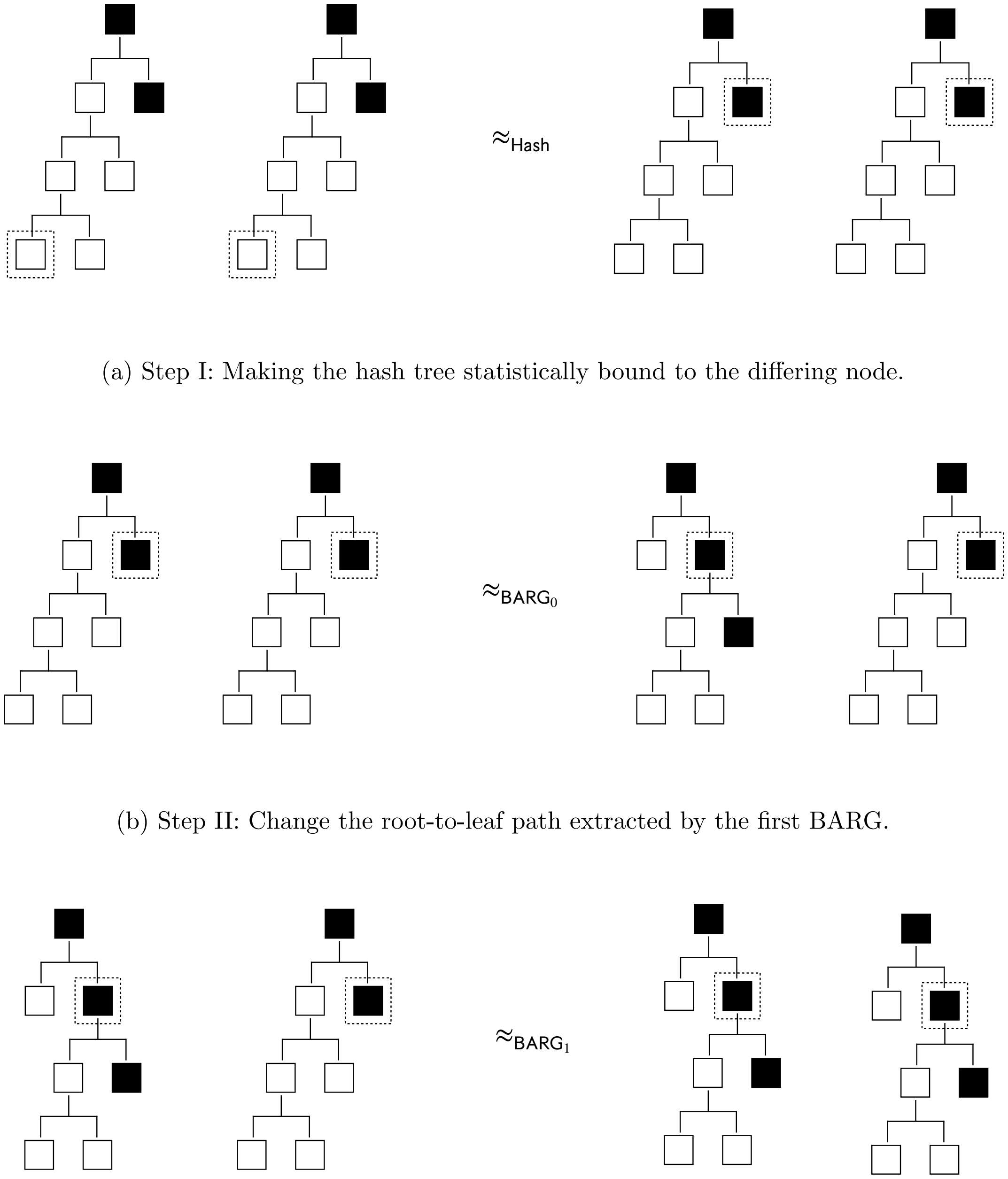
$\mathsf{Hash}(y_1, \ldots, y_n)$. By the extractability of the hash function Hash, this can be implemented efficiently, given a trapdoor. The effect of this step is to "clean up" the output of $\mathsf{Hash}(y_1, \ldots, y_n)$ by adversarial $y_{j \neq i}$. Next, (ii) we open up the construction of compute-and-compare obfuscation [WZ17, GKW17] to show that, if $f(x)$ has high entropy, then $\tilde{C}(x)$ is statistically close to uniform. In other words, $\tilde{C}$ is a good randomness extractor. This allows us to conclude that the output of the aggregate is pseudorandom.

Aggregate Uniqueness via Proof Switching. We show that it is hard to compute valid proofs for an aggregate $y^* \neq \tilde{y}$, where $\tilde{y}$ is honestly computed. For the case of our construction, this means that the two aggregate must correspond to two different roots of the hash tree. Naively, one could hope to reduce this property directly to the underlying VRF, by extracting two valid proofs π_i and π_i^* for two different images y_i and y_i^*. However, the fact that we have two different roots, does not mean that we have a mismatch for exactly the i-th leaf! We have to worry about the fact that the BARG extraction may "miss" the leaf where the mismatch starts. Our strategy to rule this out is to track down the mismatch by iteratively moving towards it, but alternating between the two BARGs, to make sure that we can still extract from one, while arguing indistinguishability for the other. A pictorial description of the process is given in Fig. 1.

In more details, say that we set both BARGs to be binding for the left-most leaf. This allows us to extract two root-to-leaf paths (path_0 and path_1) which, for the sake of this overview, we always assume to be identical (it can be shown by a reduction to collision-resistance that this condition cannot be violated). As we discussed earlier, these root-to-leaf path must lead to a different root than the honestly computed one. Our first observation is that the extracted root-to-leaf paths may be identical to the one in the honest tree in the leafs, but they must diverge at some point (as otherwise they would result in the same root). Let N be the first differing node; in the first hybrid, we make the hash function Hash binding to N. This allows us to freely change the index of the two BARGs without worrying about this node having a different value, since it is now statistically bound to the root. We then change the first BARG to be binding on the left-most leaf in the sub-tree spanned by N, and we can use the other BARG to extract the path while we argue indistinguishability. We then switch the roles of the BARGs and apply this operation again. At this point the node N must still differ from the honestly computed one, however it must also be the case that one of the *children* of N must differ from the honest root-to-leaf path! That is, this argument has allowed us to move one level lower in the tree. Applying this argument iteratively, we will eventually reach the case where the differing node is a leaf, in which case we can appeal to the uniqueness of the base (non-aggregate) VRF, and conclude our proof.

Application: Aggregate Proof of Stake. To exemplify the applicability of aggregate VRFs, we briefly discuss how VRF can be used in proof-of-stake (PoS) protocols to reduce the storage cost by aggregating VRF proofs. A protocol based on similar principles was recently proposed in [FHASW23], and we recall here the main ideas. In existing PoS protocols, such as [GHM$^+$17, DGKR18], users upload their VRF verification key to the blockchain. In each epoch, each user can evaluate the VRF on the slot number (which will play the role as a unique identifier for the epoch) to compute the corresponding image. If the image is less than (a function of) their wealth, then that user is identified as the designated party to generate the new block. Such party can then compute a proof to convince all the other parties that the VRF was evaluated correctly. This way, it can be publicly verified that that the user was indeed the winner of the round.

Using a VRF that has aggregation properties, one can compress all VRF proofs into a *single one*, without affecting the ability of the participants to verify that the blockchain was correctly



(a) Step I: Making the hash tree statistically bound to the differing node.

(b) Step II: Change the root-to-leaf path extracted by the first BARG.

(c) Step III: Change the root-to-leaf path extracted by the second BARG.

Figure 1: Example of a step of the extraction procedure for a small tree. Black nodes indicate nodes where the two extracted paths (represented in the picture) differ from the honest path. A dotted line around a node indicates that the hash function is statistically bound to that node.

computed: By the aggregate uniqueness property, it is hard for any adversary to compute an aggregate proof for a false set of values. This means that there is no need to store a VRF proof for each epoch, instead this information can be compressed into a *single group element*.

1.3 Related Work

As mentioned before, VRF is a well-studied topic in cryptography, both from a theoretical and practical angle. Thus, many constructions [DY05, HJ16, Lys02, Jag15, HW10, GHKW17, Bit17, GLOW21, Koh19, EKS$^+$21, BDE$^+$22, ESLR23] have emerged over the years. However, to the best of our knowledge, no scheme that satisfies key-homomorphism (as defined in this work) was known prior to our work. In [LBM20] the notion of aggregation is considered in the context of VRF, except that it is in the *secret key settings*, i.e., the aggregation algorithm requires the knowledge of the secret key of the VRF. This is in contrast with our work, where aggregation is a public procedure. The notion of publicly aggregate VRF was concurrently and independently studied in [DDKL24], where a construction from bilinear parings is also proposed. Their scheme is quite different from the one we present, and in particular it does not satisfy the key-homomorphic property (as defined in our work). On the other hand, their construction satisfies the stronger notions of composable security and forward secrecy.

We also mention that the notion of key-homomorphic VRF was discussed (but not formalized) in [FHASW23] although the existence of a construction was left as an open problem. In their work, they implicitly construct a key-homomorphic VRF with a small domain and a small co-domain (i.e., the VRF is only defined on a polynomial number of inputs), which suffices for their application.

A related notion is that of verifiable unpredictable functions (VUF), which offers the weaker property of unpredictability, as opposed to pseudorandomness. Although a VUF can be generically transformed into a VRF, e.g., by hashing the image with a random oracle, this transformation does not preserve key homomorphism. Therefore, a key-homomorphic VUF does not immediately imply a key-homomorphic VRF. We also mention that aggregate VUF have been studied in the literature [BGLS03, KM13], but their definitions are not applicable to the notion of VRF, due to the weaker security requirement. Besides the definitional mismatch, building aggregate VRFs is much more complex than the case of VUF, since aggregation must preserve (i) the pseudorandomness and the (ii) uniqueness of the aggregate, whereas none of these properties is relevant in the context of VUFs/signatures. Finally, we mention that key-homomorphic PRFs have been studied [BLMR13], and we view our work as a continuation of this line of research, answering a natural question on adding verifiability to key-homomorphic primitives.

1.4 Open Problems

We hope that our work will motivate researchers to further study this cryptographic object, both from a foundational and practical perspective. For instance, it would be interesting to find a construction of key-homomorphic VRFs secure in the standard model, or from post-quantum assumptions. Furthermore, on the theoretical side, an interesting question is whether one can construct a key-homomorphic/aggregate VRF (perhaps satisfying weaker security notions) in the plain model, i.e., without any trusted setup. We also leave open the formalization of the security of aggregate VRFs in the presence of a malicious aggregator. On the other end of the spectrum, an interesting open question is finding more applications of this primitive, and rigorously analyze its security in the context of more complex protocols.

2 Preliminaries

Throughout this work, we write λ to denote the security parameter. We say a function f is negligible in the security parameter λ if $f = \lambda^{-\omega(1)}$. We say an algorithm is efficient if it runs in probabilistic polynomial time (PPT) in the length of its input. For a distribution D, we write $x \leftarrow\$ S$ to denote that x is sampled uniformly at random from D. We say that two distributions D_0 and D_1 are computationally indistinguishable if there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$ and for all PPT algorithms $\mathcal{A}$ it holds that

$$|\Pr[1 \leftarrow \mathcal{A}(x) : x \leftarrow\$ D_0] - \Pr[1 \leftarrow \mathcal{A}(x) : x \leftarrow\$ D_1]| = \mu(\lambda).$$

2.1 Bilinear Groups

Let $\mathcal{G} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow\$ \mathsf{GroupGen}(1^\lambda)$ be a generator of an asymmetric bilinear group generated by g_1 and g_2 of prime order p, with an efficiently computable pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. We also assume that it is efficient to test whether a given element g is a member of the group $\mathbb{G}_1$, which is sometimes referred to as the *certified group* settings, see e.g. [Sco21] for details. We recall the following variant of the Bilinear Diffie-Hellman (BDH) assumption.

Definition 2.1 (BDH Assumption)**.** We say that a bilinear group generator $\mathsf{GroupGen}$ is BDH-hard, if the following distributions are computationally indistinguishable

$$\left(g_1, g_2, g_1^a, g_1^b, g_2^a, g_2^b, g_2^c, e(g_1, g_2)^{\frac{bc}{a}}\right) \approx \left(g_1, g_2, g_1^a, g_1^b, g_2^a, g_2^b, g_2^c, e(g_1, g_2)^r\right)$$

where $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow\$ \mathsf{GroupGen}(1^\lambda)$ and $(a, b, c, r) \leftarrow\$ \mathbb{Z}_p^4$.

We also define two less standard variants of this assumption, where the adversary is given additional extra group elements in its view. In App. A, we show that both assumptions hold unconditionally in the generic group model (GGM).

Definition 2.2 (Augmented BDH Assumption)**.** We say that a bilinear group generator $\mathsf{GroupGen}$ is ABDH-hard, if the following distributions are computationally indistinguishable

$$\left(g_1, g_2, g_1^a, g_1^b, g_2^a, g_2^b, g_2^c, e(g_1, g_2)^{\frac{c}{a}}, e(g_1, g_2)^{\frac{c^2}{a}}, e(g_1, g_2)^{\frac{bc}{a}}\right)$$
$$\approx \left(g_1, g_2, g_1^a, g_1^b, g_2^a, g_2^b, g_2^c, e(g_1, g_2)^{\frac{c}{a}}, e(g_1, g_2)^{\frac{c^2}{a}}, e(g_1, g_2)^r\right)$$

where $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow\$ \mathsf{GroupGen}(1^\lambda)$ and $(a, b, c, r) \leftarrow\$ \mathbb{Z}_p^4$.

Definition 2.3 (External-Inverted BDH Assumption)**.** We say that a bilinear group generator $\mathsf{GroupGen}$ is XIBDH-hard, if the following distributions are computationally indistinguishable

$$\left(g_1, g_2, g_1^a, g_1^{ab}, g_1^{ac}, g_1^{b/c}, g_2^d, g_2^{dc}\right) \approx \left(g_1, g_2, g_1^a, g_1^{ab}, g_1^{ac}, g_1^r, g_2^d, g_2^{dc}\right)$$

where $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow\$ \mathsf{GroupGen}(1^\lambda)$ and $(a, b, c, d, r) \leftarrow\$ \mathbb{Z}_p^5$.

3 Definitions

We recall the notion of a verifiable random function (VRF) [MRV99]. Informally, a VRF is a keyed function

$$\mathrm{VRF} : \mathcal{K} \times \{0,1\}^\lambda \to \mathcal{Y}$$

that behaves like a random function, and simultaneously allows one to verify the validity of its outputs.

3.1 Verifiable Random Functions

We formally define the notion of a VRF, we adopt the standard definition [MRV99], except for two syntactical modifications: First, we require the verification key to be a deterministic and *bijective* function of the secret key. This is needed for our definition of aggregate pseudorandomness (Def. 3.6). Furthermore, we allow for a one-time trusted setup.

Definition 3.1 (VRF)**.** A verifiable random function (VRF) is a tuple of polynomial-time algorithms with the following syntax.

- $\mathsf{Setup}(1^\lambda)$: On input the security parameter 1^λ, the setup algorithm returns the common reference string crs.
- $\mathsf{Gen}(\mathsf{crs})$: On input the common reference string crs, the generation algorithm samples a secret key k and returns $k \in \mathcal{K}$ and the verification key $\mathsf{vk} = \mathsf{VKey}(k)$, where VKey is a bijective function.
- $\mathsf{Eval}(k, x)$: On input the secret key $k \in \mathcal{K}$ and some point $x \in \{0,1\}^\lambda$, the evaluation algorithm returns an image $y \in \mathcal{Y}$.
- $\mathsf{Prove}(k, x)$: On input the secret key $k \in \mathcal{K}$ and some point $x \in \{0,1\}^\lambda$, the prover algorithm returns a proof $\pi \in \mathcal{P}$.
- $\mathsf{Verify}(\mathsf{vk}, x, y, \pi)$: On input the verification key vk, some point $x \in \{0,1\}^\lambda$, an image $y \in \mathcal{Y}$, and a proof $\pi \in \mathcal{P}$, the verification algorithm returns a bit $b \in \{0,1\}$ denoting acceptance or rejection.

For correctness, we require that for all $\lambda \in \mathbb{N}$, all crs in the support of $\mathsf{Setup}(1^\lambda)$, all key pairs (k, vk) in the support of the $\mathsf{Gen}(\mathsf{crs})$ algorithm, and all points $x \in \{0,1\}^\lambda$ it holds that

$$\mathsf{Verify}(\mathsf{vk}, x, \mathsf{Eval}(k, x), \mathsf{Prove}(k, x)) = 1.$$

Next, we recall the security requirements of a VRF. The first property demands that it should be impossible to find a valid proof π^* for an invalid image y^*.

Definition 3.2 (Uniqueness)**.** A VRF satisfies *uniqueness* if for $\lambda \in \mathbb{N}$, all crs in the support of $\mathsf{Setup}(1^\lambda)$, all verification keys vk, all points $x \in \{0,1\}^\lambda$, and all images $(y_0, y_1) \in \mathcal{Y}^2$ and proofs $(\pi_0, \pi_1) \in \mathcal{P}^2$ it holds that

$$1 = \mathsf{Verify}(\mathsf{vk}, x, y_0, \pi_0) = \mathsf{Verify}(\mathsf{vk}, x, y_1, \pi_1) \implies y_0 = y_1.$$

Finally, we recall the standard notion of pseudorandomness, which requires that the adversary should not be able to distinguish the output of a VRF from a uniformly sampled function, even given access to an oracle that computes images an proofs on arbitrary points chosen by the distinguisher.

Definition 3.3 (Pseudorandomness)**.** A VRF satisfies *pseudorandomness* if there exists a negligible function μ such that for $\lambda \in \mathbb{N}$ and all admissible PPT adversaries $\mathcal{A}$ it holds that

$$\left| \frac{1}{2} - \Pr\left[1 \leftarrow \mathsf{ExpRandom}_{\mathcal{A}}(1^{\lambda})\right] \right| = \mu(\lambda)$$

where the experiment $\mathsf{ExpRandom}_{\mathcal{A}}$ is defined as follows.

- The challenger samples a reference string $\mathsf{crs} \leftarrow\!\$\; \mathsf{Setup}(1^{\lambda})$ and a key pair $(k, \mathsf{vk}) \leftarrow\!\$\; \mathsf{Gen}(\mathsf{crs})$ and sends $(\mathsf{crs}, \mathsf{vk})$ to $\mathcal{A}$.

- $\mathcal{A}$ can query adaptively and at any time an oracle on input x_i and receives back from the challenger a pair (y_i, π_i) where $y_i \leftarrow \mathsf{Eval}(k, x_i)$ and $\pi_i \leftarrow \mathsf{Prove}(k, x_i)$.

- At any point (including between evaluation queries) $\mathcal{A}$ can query a challenge input x^*. The challenger flips a coin $b \leftarrow\!\$\; \{0, 1\}$ and computes

$$\begin{cases} y^* \leftarrow \mathsf{Eval}(k, x^*) & \text{if } b = 0 \\ y^* \leftarrow\!\$\; \mathcal{Y} & \text{if } b = 1 \end{cases}.$$

- In the end of the experiment, $\mathcal{A}$ outputs a guess b^*. The experiment returns 1 if and only if $b^* = b$.

Furthermore, we say that $\mathcal{A}$ is admissible if $x^* \notin Q$, where Q denotes the set of points queried to the oracle as defined above.

3.2 Key Homomorphism

We define the central object of this work, namely the notion of *key-homomorphic* VRF (KH-VRF). A KH-VRF is defined like a standard VRF with some additional structural properties, which allows one to meaningfully combine the images and the proofs computed under different keys. To formally define this notion, we will endow the key, the image, and the proof domain with a group structure, which allows us to state this additional property.

Definition 3.4 (Key Homomorphism)**.** A VRF is *key homomorphic* if $(\mathcal{K}, \oplus)$, $(\mathcal{Y}, \otimes)$, $(\mathcal{P}, \otimes)$ are groups and for all $(k_0, k_1) \in \mathcal{K}^2$ and all $x \in \{0, 1\}^{\lambda}$ it holds that

- $\mathsf{Eval}(k_0, x) \otimes \mathsf{Eval}(k_1, x) = \mathsf{Eval}(k_0 \oplus k_1, x)$, and

- $\mathsf{Prove}(k_0, x) \otimes \mathsf{Prove}(k_1, x) = \mathsf{Prove}(k_0 \oplus k_1, x)$.

Looking ahead at our instantiation in Sec. 4, the group $(\mathcal{K}, \oplus)$ will consist of the additive group $\mathbb{Z}_p$, for some prime p, whereas $(\mathcal{Y}, \otimes)$ and $(\mathcal{P}, \otimes)$ will be two multiplicative groups of order p.

3.3 Aggregation

Given the above defined homomorphic property, it is natural to derive an aggregation Agg algorithm that allows us to compress n VRFs into a single one, while at the same time outputting a proof of validity for the aggregate. We formally define the correctness of the algorithm in the following. Importantly, we define the algorithms to be *deterministic* so that running the same aggregation twice leads to exactly the same output.

Definition 3.5 (Aggregation)**.** A VRF is *aggregatable* if there exists a triple of deterministic polynomial-time algorithms with the following syntax.

- $\mathsf{Agg}(x, \{\mathsf{vk}_i, y_i\}_{i=1\dots n})$: On input a point $x \in \{0,1\}^\lambda$, n verification keys $(\mathsf{vk}_1, \dots, \mathsf{vk}_n)$ and images $(y_1, \dots, y_n) \in \mathcal{Y}^n$, the aggregation algorithm outputs an aggregate key $\tilde{\mathsf{vk}}$ and an aggregate image $\tilde{y}$.
- $\mathsf{AggProve}(x, \{\mathsf{vk}_i, \pi_i\}_{i=1\dots n})$: On input a point $x \in \{0,1\}^\lambda$, n verification keys $(\mathsf{vk}_1, \dots, \mathsf{vk}_n)$, and proofs $(\pi_1, \dots, \pi_n) \in \mathcal{P}^n$, the aggregate proof algorithm outputs an aggregate proof $\tilde{\pi}$.
- $\mathsf{AggVerify}(\{\mathsf{vk}_i\}_{i=1\dots n}, x, \tilde{y}, \tilde{\pi})$: On input n verification keys $(\mathsf{vk}_1, \dots, \mathsf{vk}_n)$, a point $x \in \{0,1\}^\lambda$, and an aggregate image-proof pair $(\tilde{y}, \tilde{\pi})$ the aggregate verification algorithm outputs a bit $b \in \{0,1\}$ denoting acceptance or rejection.

The correctness of the aggregation algorithm requires that the aggregated image-proof pair correctly verifies, provided that the inputs are all correctly formed. More formally, we require that for all $\lambda \in \mathbb{N}$, all polynomials $n = n(\lambda)$, all crs in the support of $\mathsf{Setup}(1^\lambda)$, all key pairs (k_i, vk_i) in the support of the $\mathsf{Gen}(\mathsf{crs})$ algorithm, and all points $x \in \{0,1\}^\lambda$ it holds that

$$\mathsf{AggVerify}\left(\{vk_i\}_{i=1\dots n}, x, \tilde{y}, \tilde{\pi}\right) = 1,$$

where

$$(\tilde{\mathsf{vk}}, \tilde{y}) \leftarrow \mathsf{Agg}\left(x, \{\mathsf{vk}_i, \mathsf{Eval}(k_i, x)\}_{i=1\dots n}\right) \quad \text{and} \quad \tilde{\pi} \leftarrow \mathsf{AggProve}\left(x, \{\mathsf{vk}_i, \mathsf{Prove}(k_i, x)\}_{i=1\dots n}\right).$$

Next we present formal definitions for the security of aggregate VRFs. In all of our definitions, we model the aggregation process as performed by an honest party, and therefore we always assume that the aggregation algorithms are run *honestly.* We leave the formalization of security in the presence of a malicious aggregator as ground for future work.

Aggregate Pseudorandomness. A desirable property for an aggregate VRF is that the output of the aggregation algorithm should still satisfy pseudorandomness, even if some of the keys are controlled by the adversary. It turns out that formalizing the correct version of this property is surprisingly subtle.

To understand our definitional choice, consider the following natural (but flawed) attempt at a formal definition: Given the adversary oracle access to the challenge VRF, we allow the adversary to additionally return some verification keys of its choice $\{\mathsf{vk}_i\}_{i=1\dots n}$, along with the corresponding image-proof pairs $\{y_i, \pi_i\}_{i=1\dots n}$. The challenger then checks whether the proofs verify, and if so it computes either

$$y^* \leftarrow \mathsf{Agg}\left(x, \mathsf{vk}, y, \{\mathsf{vk}_i, y_i\}_{i=1\dots n}\right) \quad \text{or} \quad y^* \leftarrow_\$ \mathcal{Y}.$$

While this is a perfectly well-defined notion, we argue that this definition is *too weak*, since it does not allow the adversary to set its key depending on the honest keys. To see why this is the case, consider the simplified case where the aggregation of images is simply their sum $\tilde{y} = \sum_i y_i$. The adversary may launch an *adaptive attack* by setting its own verification key vk^* depending on the verification key of the challenger vk, in such a way that $y^* = -y$. This way, the contribution of y to the aggregate $\tilde{y}$ is canceled out, and the output of the aggregation algorithm is perfectly predictable by the adversary, even before seeing the answer to the challenge query. Nevertheless, this attacker would not be able to win the experiment as outlined above, because it would not be able to compute y^* *before* seeing y, and thus cannot send it as part of the query to the challenge oracle. This is a strong indication that one needs a stronger definition of security for aggregate pseudorandomness.

To obviate this problem, our idea is to make the life of the attacker easier, by simply delegating to the challenger the task of computing the image of a verification key sent by the adversary. In general, this is not an efficient procedure, but we can bypass this obstacle by letting the challenger run in unbounded time. Note that the output of this procedure is always well-defined, since the function VKey is bijective, and therefore its inverse is uniquely determined. We present the formal definition below.

Definition 3.6 (Aggregate Pseudorandomness)**.** A VRF satisfies *aggregate pseudorandomness* if there exists a negligible function μ such that for $\lambda \in \mathbb{N}$ and all admissible PPT adversaries $\mathcal{A}$ it holds that

$$\left| \frac{1}{2} - \Pr\left[1 \leftarrow \mathsf{ExpAggRandom}_{\mathcal{A}}(1^\lambda)\right] \right| = \mu(\lambda)$$

where the experiment $\mathsf{ExpAggRandom}_{\mathcal{A}}$ is defined as follows.

- The challenger samples a reference string $\mathsf{crs} \leftarrow\$ \mathsf{Setup}(1^\lambda)$ and a key pair $(k, \mathsf{vk}) \leftarrow\$ \mathsf{Gen}(\mathsf{crs})$ and sends $(\mathsf{crs}, \mathsf{vk})$ to $\mathcal{A}$.
- $\mathcal{A}$ can query adaptively and at any time an oracle on input x_i and receives back from the challenger a pair (y_i, π_i) where $y_i \leftarrow \mathsf{Eval}(k, x_i)$ and $\pi_i \leftarrow \mathsf{Prove}(k, x_i)$.
- At any point (including between evaluation queries) $\mathcal{A}$ can query a challenge input x^* along with some challenge keys $\{\mathsf{vk}_i\}_{i=1\dots n}$. The challenger (inefficiently) computes $k_i \leftarrow \mathsf{VKey}^{-1}(\mathsf{vk}_i)$ and sets $y_i \leftarrow \mathsf{Eval}(k_i, x^*)$. Then it flips a coin $b \leftarrow\$ \{0,1\}$ and computes

$$\begin{cases} y^* \leftarrow \mathsf{Agg}\left(x^*, \mathsf{vk}, \mathsf{Eval}(k, x^*), \{\mathsf{vk}_i, y_i\}_{i=1\dots n}\right) & \text{if } b = 0 \\ y^* \leftarrow\$ \mathcal{Y} & \text{if } b = 1 \end{cases}.$$

- In the end of the experiment, $\mathcal{A}$ outputs a guess b^*. The experiment returns 1 if and only if $b^* = b$.

Furthermore, we say that $\mathcal{A}$ is admissible if $x^* \notin Q$, where Q denotes the set of points queried to the oracle as defined above.

Aggregate Binding. We also require that no attacker should be able to find two different preimages for a given aggregate. This property is useful in settings where one stores an aggregate proof, and wants to prevent changes the corresponding images after the fact. Given that the size of the aggregate is independent of n, we can only hope to achieve a computational guarantee, where finding collision is only computationally hard. We present a formal definition below. We remark that, since the aggregation algorithm is deterministic, there is no need to provide the aggregate as an input to the adversary explicitly, since it can be recomputed given the information that the adversary already possesses.

Definition 3.7 (Aggregate Binding)**.** A VRF satisfies *aggregate binding* if there exists a negligible function μ such that for $\lambda \in \mathbb{N}$ and all PPT adversaries $\mathcal{A}$ it holds that

$$\Pr\left[1 \leftarrow \mathsf{ExpAggBind}_{\mathcal{A}}(1^\lambda)\right] = \mu(\lambda)$$

where the experiment $\mathsf{ExpAggBind}_{\mathcal{A}}$ is defined as follows.

- The challenger samples a reference string $\mathsf{crs} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$ and sends it to $\mathcal{A}$.
- $\mathcal{A}$ returns a point x^*, n verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$, two tuples $(y_1, \ldots, y_n)$ and $(y_1^*, \ldots, y_n^*)$, and proofs $(\pi_1, \ldots, \pi_n)$.
- The challenger computes

  $$\tilde{\pi} \leftarrow \mathsf{AggProve}\left(x^*, \{\mathsf{vk}_i, \pi_i\}_{i=1\ldots n}\right) \quad \text{and} \quad \tilde{y} \leftarrow \mathsf{Agg}\left(x^*, \{\mathsf{vk}_i, y_i^*\}_{i=1\ldots n}\right)$$

  and outputs 1 if and only if:

  $$\left\{\mathsf{Verify}(\mathsf{vk}_i, x^*, y_i, \pi_i) \overset{?}{=} 1\right\}_{i=1\ldots n} \quad \text{and} \quad \mathsf{AggVerify}\left(\{vk_i\}_{i=1\ldots n}, x, \tilde{y}, \tilde{\pi}\right) \overset{?}{=} 1$$

  and furthermore $(y_1, \ldots, y_n) \neq (y_1^*, \ldots, y_n^*)$.

Aggregate Uniqueness. Finally, we require that an aggregate key must also be a valid VRF key, and in particular it must satisfy uniqueness. We formalize this property below.

Definition 3.8 (Aggregate Uniqueness)**.** A VRF satisfies *aggregate uniqueness* if for $\lambda \in \mathbb{N}$, all polynomials $n = n(\lambda)$, all crs in the support of $\mathsf{Setup}(1^\lambda)$, all verification keys vk_i, and all points $x \in \{0,1\}^\lambda$, it holds that $\tilde{\mathsf{vk}}$ satisfies uniqueness (Def. 3.2), where

$$(\tilde{\mathsf{vk}}, \tilde{y}) \leftarrow \mathsf{Agg}\left(x, \{\mathsf{vk}_i, \mathsf{Eval}(k_i, x)\}_{i=1\ldots n}\right).$$

4 Key-Homomorphic VRFs from Bilinear Groups

We describe our main construction for a KH-VRF. We begin by introducing a simple scheme with key-homomorphic properties, then we show how to extend it to achieve several extra properties.

4.1 Base Scheme

We present our base KH-VRF in the following. The construction assumes the existence of a hash function $\mathcal{H}$ where

$$\mathcal{H} : \{0,1\}^\lambda \rightarrow \mathbb{G}_2$$

is modelled as a random oracle. The algorithms are specified below.

- $\mathsf{Setup}(1^\lambda)$: Sample a bilinear group
$$(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow\$ \ \mathsf{GroupGen}(1^\lambda)$$
along with an integer $s \leftarrow\$ \ \mathbb{Z}_p$. Then compute $S = g_1^s$ and set the crs to (g_1, g_2, S).
- $\mathsf{Gen}(\mathsf{crs})$: Sample $k \leftarrow\$ \ \mathbb{Z}_p$ and set vk to S^k. Return (k, vk).
- $\mathsf{Eval}(k, x)$: Compute and output
$$y = e\left(g_1, \mathcal{H}(x)\right)^k.$$
- $\mathsf{Prove}(k, x)$: Compute and output $\pi = \mathcal{H}(x)^k$.
- $\mathsf{Verify}(\mathsf{vk}, x, y, \pi)$: Accept if $\mathsf{vk} \in \mathbb{G}_1$ and
$$e\left(\mathsf{vk}, \mathcal{H}(x)\right) \stackrel{?}{=} e\left(S, \pi\right) \quad \text{and} \quad y \stackrel{?}{=} e\left(g_1, \pi\right).$$

We remark that, although we described the setup algorithm as sampling the secret integer s explicitly, an alternative (but equivalent) sampling procedure would be to derive S by hashing into the group $\mathbb{G}_1$. This way, the setup can be done in a completely *transparent* manner, i.e., using public random coins.

It is easy to check that the scheme is correct by simply substituting the variables
$$e\left(\mathsf{vk}, \mathcal{H}(x)\right) = e\left(g_1, \mathcal{H}(x)\right)^{ks} = e\left(g_1^s, \mathcal{H}(x)^k\right) = e\left(S, \pi\right)$$
and
$$y = e\left(g_1, \mathcal{H}(x)\right)^k = e\left(g_1, \mathcal{H}(x)^k\right) = e\left(g_1, \pi\right).$$
To establish that the scheme is also key-homomorphic, it is enough to observe that for all polynomials $n = n(\lambda)$ it holds that
$$\prod_{i=1}^n \mathsf{Eval}(k_i, x) = \prod_{i=1}^n e\left(g_1, \mathcal{H}(x)\right)^{k_i} = e\left(g_1, \mathcal{H}(x)\right)^{\sum_{i=1}^n k_i} = \mathsf{Eval}\left(\sum_{i=1}^n k_i, x\right)$$
and similarly
$$\prod_{i=1}^n \mathsf{Prove}(k_i, x) = \prod_{i=1}^n \mathcal{H}(x)^{k_i} = \mathcal{H}(x)^{\sum_{i=1}^n k_i} = \mathsf{Prove}\left(\sum_{i=1}^n k_i, x\right).$$
Furthermore, the scheme satisfies uniqueness, since π is uniquely determined by S, vk, and x.

Analysis. We state the main theorem of this section, namely that the base scheme is secure if the BDH assumption holds.

Theorem 4.1. If the group generator $\mathsf{GroupGen}$ is BDH-hard, then the construction as described above satisfies pseudorandomness, in the random oracle model.

Proof. The pseudorandomness of our VRF follows from a reduction to the BDH assumption (Def. 2.1). Recall that the assumption stipulates that the following distributions are computationally indistinguishable

$$\left(g_1, g_2, g_1^a, g_1^b, g_2^a, g_2^b, g_2^c, e(g_1, g_2)^{\frac{bc}{a}}\right) \approx \left(g_1, g_2, g_1^a, g_1^b, g_2^a, g_2^b, g_2^c, e(g_1, g_2)^r\right)$$

where $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow\$ \ \mathsf{GroupGen}(1^\lambda)$ and $(a, b, c, r) \leftarrow\$ \ \mathbb{Z}_p^4$. By a variable renaming, this is equivalent to distinguishing between

$$\left(g_1, g_2, g_1^s, g_1^{ks}, g_2^s, g_2^{ks}, g_2^z, e(g_1, g_2)^{kz}\right) \approx \left(g_1, g_2, g_1^s, g_1^{ks}, g_2^s, g_2^{ks}, g_2^z, e(g_1, g_2)^r\right).$$

On input a sample from either distribution, the reduction sets $S = g_1^s$ and $\mathsf{vk} = g_1^{ks}$, then it activates the adversary $\mathcal{A}$. Let Q be the number of queries of $\mathcal{A}$ to the random oracle, the reduction guesses an index $i^* \in \{1, \ldots, Q\}$ and simulates the oracle as follows: For $i \neq i^*$, on input x_i the reduction programs the random oracle to

$$\mathcal{H}(x_i) = (g_2^s)^{\rho_i} \text{ where } \rho_i \leftarrow\$ \ \mathbb{Z}_p.$$

On the other hand, on input x_{i^*} the reduction sets

$$\mathcal{H}(x_{i^*}) = g_2^z.$$

Next, we show how the reduction simulates the queries to the VRF evaluation oracle. Without loss of generality, we assume that the adversary already queried the input x_i to the random oracle (otherwise, the reduction can perform the query itself before answering to $\mathcal{A}$). If $x_i = x_{i^*}$, the reduction aborts and returns a random bit. Otherwise, the reduction computes

$$\pi_i = \left(g_2^{ks}\right)^{\rho_i} \quad \text{and} \quad y_i = e(g_1, \pi)$$

by fetching the appropriate value of ρ_i. On input the challenge query x^*, the reduction checks if $x^* = x_{i^*}$ and returns a random bit if this is not the case. Otherwise, it computes

$$y^* = \left\{e(g_1, g_2)^{kz}, e(g_1, g_2)^r\right\}$$

i.e., it simply returns the last group element from the challenge tuple. The reduction returns whatever bit $\mathcal{A}$ returns.

First, observe that the reduction is efficient, as it only performs basic group operations. Consider the case where the reduction guesses correctly the query to the random oracle that contains the challenge input x^*. In this case, the reduction perfectly simulates the queries to the oracle offered by the pseudorandomness experiment, since

$$\pi_i = \left(g_2^{ks}\right)^{\rho_i} = (g_2^{s\rho_i})^k = \mathcal{H}(x_i)^k$$

and $s\rho_i$ is uniformly distributed in $\mathbb{Z}_p$. Finally, note that the case where the last group element is uniform perfectly simulates the experiment where $b = 1$, whereas in the other case

$$y^* = e(g_1, g_2)^{kz} = e(g_1, g_2^z)^k = e\left(g_1, \mathcal{H}(x^*)\right)^k$$

the view is identical for the case where $b = 0$. Since the reduction guesses correctly with probability at least $1/Q$, which is inverse polynomial, the success probability of the reduction in distinguishing the two distributions is at most a polynomial factor smaller than to that of $\mathcal{A}$. This contradicts the BDH assumption and concludes our proof. □

4.2 Aggregation

The base scheme described above enjoys useful structural properties that make it amenable to aggregation. A naive proposal to aggregate VRF would be to simply compute the product of the images and proofs, i.e.,

$$\tilde{y} = \prod_{i=1}^{n} y_i = \prod_{i=1}^{n} e\left(g_1, \mathcal{H}(x)\right)^{k_i} \quad \text{and} \quad \tilde{\pi} = \prod_{i=1}^{n} \pi_i = \prod_{i=1}^{n} \mathcal{H}(x)^{k_i}.$$

This is syntactically correct, since it corresponds to the output of the Eval and Prove algorithms on with key $\tilde{k} = \sum_{i=1}^{n} k_i$. Unfortunately this simple proposal suffers from many issues: (i) First of all, it does not achieve aggregate pseudorandomness, since it is vulnerable to *rogue key attacks.* To see why, it suffices to observe that, on input an honest key $\mathsf{vk} = S^k$, an attacker could set their key to $\mathsf{vk}^{-1} = S^{-k}$ (which is efficiently computable). This way, the output of the aggregate on any input x is simply

$$e\left(g_1, \mathcal{H}(x)\right)^{k} \cdot e\left(g_1, \mathcal{H}(x)\right)^{-k} = 1$$

which is clearly not pseudorandom. This particular problem can be fixed by adding non-interactive zero-knowledge proof (NIZK) [FLS90] to the public key, however the resulting VRF would no longer be key-homomorphic. Moreover, even with the NIZK in place, (ii) this construction does not satisfy aggregate binding. To illustrate this last point, observe that it is easy to compute an x^* and a pair $(y_1, \ldots, y_n) \neq (y_1^*, \ldots, y_n^*)$ such that

$$\mathsf{Agg}\left(x^*, \{\mathsf{vk}_i, y_i\}_{i=1\ldots n}\right) = \prod_{i=1}^{n} y_i = \prod_{i=1}^{n} y_i^* = \mathsf{Agg}\left(x^*, \{\mathsf{vk}_i, y_i^*\}_{i=1\ldots n}\right)$$

by simple linear algebra. In the following, we propose a different aggregation function that simultaneously resolves both limitations.

Aggregate VRF. Our approach is inspired by the work of [BDN18], although our construction and analysis turns out to be substantially different. We assume the existence of two hash functions

$$\tilde{\mathcal{G}} : \{0,1\}^\lambda \times \mathbb{G}_1^n \times \mathbb{G}_T^n \to \{0,1\}^\lambda \quad \text{and} \quad \tilde{\mathcal{H}} : \{0,1\}^\lambda \to \mathbb{Z}_q^n.$$

modeled as random oracles. Loosely speaking, the first random oracle will be used to sample a random seed, which is included in the proof, whereas the second random oracle will be used to sample the coefficients for a linear combination of the VRF images. Since the output of this oracle is unpredictable to the adversary, we will be able to show that no (efficient) adversary can cancel out the contributions of the honest keys. Our aggregation algorithms are described below.

- $\mathsf{Agg}(x, \{\mathsf{vk}_i, y_i\}_{i=1\ldots n})$: Compute $R \leftarrow \tilde{\mathcal{G}}(x, \mathsf{vk}_1, \ldots, \mathsf{vk}_n, y_1, \ldots, y_n)$ and $(r_1, \ldots, r_n) \leftarrow \tilde{\mathcal{H}}(R)$, then return $\tilde{y} = \prod_{i=1}^{n} y_i^{r_i}$.
- $\mathsf{AggProve}(x, \{\mathsf{vk}_i, \pi_i\}_{i=1\ldots n})$: Compute $R \leftarrow \tilde{\mathcal{G}}(x, \mathsf{vk}_1, \ldots, \mathsf{vk}_n, e(g_1, \pi_1), \ldots, e(g_1, \pi_n))$ along with $(r_1, \ldots, r_n) \leftarrow \tilde{\mathcal{H}}(R)$, then return $(\tilde{\pi} = \prod_{i=1}^{n} \pi_i^{r_i}, R)$.
- $\mathsf{AggVerify}(\{\mathsf{vk}_i\}_{i=1\ldots n}, x, \tilde{y}, (\tilde{\pi}, R))$: Compute $(r_1, \ldots, r_n) \leftarrow \tilde{\mathcal{H}}(R)$ and define $\tilde{\mathsf{vk}} = \prod_{i=1}^{n} \mathsf{vk}_i^{r_i}$. Return

$$1 \overset{?}{=} \mathsf{Verify}(\tilde{\mathsf{vk}}, x, \tilde{y}, \tilde{\pi}) \quad \text{and} \quad y \overset{?}{\neq} 1.$$

Aggregate correctness follows immediately from the key homomorphic property of our VRF, since the pair $(\tilde{y}, \tilde{\pi})$ is a valid image-proof pair under the key $\tilde{k} = \sum_{i=1}^{n} k_i \cdot r_i$, and furthermore $y_i = e(g_1, \pi_i)$. Furthermore, $y = 1$ only with negligible probability. Next we show that the construction satisfies aggregate pseudorandomness.

The Algebraic Group Model. Before delving into the proof, we recall the basics of the algebraic group model (AGM) [FKL18]. In the AGM, all algorithms are treated as algebraic algorithms. That is, along with every group element that they return, they are required to add an *explanation* of such element as a linear combination of the input group elements. We define the notion of algebraic algorithms below.

Definition 4.2 (Algebraic Algorithms)**.** An algorithm $\mathcal{A}$ is *algebraic* if, for every group element $h \in \mathbb{G}$ that it outputs, it also returns the corresponding list of algebraic coefficients. That is

$$(h, c_1, \ldots, c_t) \leftarrow\$ \mathcal{A}(h_1, \ldots, h_t) \text{ such that } h = \prod_{i=1}^{t} h_i^{c_i}.$$

For an adversary $\mathcal{A}$ that has access to oracles during its runtime, we impose the above restriction to all group elements that it sends to the oracle. Similarly, all group elements that A receives from the oracle are treated as new inputs to $\mathcal{A}$.

We also recall the Schwartz–Zippel lemma, which is going to be useful in our analysis.

Lemma 4.3 (Schwartz–Zippel)**.** Let $P \in \mathbb{F}[X_1, \ldots, X_n]$ be a non-zero n-variate polynomial over a field $\mathbb{F}$ of degree d. Let $r_1, \ldots, r_n$ be selected uniformly at random from $\mathbb{F}$, then

$$\Pr\left[P(r_1, \ldots, r_n) = 0\right] \leq \frac{d}{|\mathbb{F}|}.$$

Aggregate Pseudorandomness. We are now in the position to show that our construction satisfies aggregate pseudorandomness in the AGM and in the random oracle model.

Theorem 4.4. If the group generator $\mathsf{GroupGen}$ is ABDH-hard, then the construction as described above satisfies aggregate pseudorandomness against algebraic adversaries, in the random oracle model.

Proof. The proof of aggregate pseudorandomnes consists of a reduction against the ABDH assumption (Def. 2.2). Recall that, up to a variable renaming, the assumption states that the following distributions are computationally indistinguishable

$$\left(g_1, g_2, g_1^s, g_1^{ks}, g_2^s, g_2^{ks}, g_2^z, e(g_1, g_2)^{\frac{z}{s}}, e(g_1, g_2)^{\frac{z^2}{s}}, e(g_1, g_2)^{kz}\right)$$
$$\approx \left(g_1, g_2, g_1^s, g_1^{ks}, g_2^s, g_2^{ks}, g_2^z, e(g_1, g_2)^{\frac{z}{s}}, e(g_1, g_2)^{\frac{z^2}{s}}, e(g_1, g_2)^{r}\right).$$

On input a sample from either distribution, the reduction sets $S = g_1^s$ and $\mathsf{vk} = g_1^{ks}$, then it activates the adversary $\mathcal{A}$. The reduction simulates the composition of random oracles $\tilde{\mathcal{H}} \circ \tilde{\mathcal{G}}$ by lazy sampling. More specifically, for any input query q to $\tilde{\mathcal{G}}$, the reduction samples a random output for $\tilde{\mathcal{H}}$ on input $\tilde{\mathcal{G}}(q)$ and records the pair $(q, \tilde{\mathcal{H}}(\tilde{\mathcal{G}}(q)))$ as the output of the combined oracle. Note that such a simulation fails when one of the outputs of $\tilde{\mathcal{G}}$ (as defined in the simulation) was queried to $\tilde{\mathcal{H}}$ before the corresponding input is queried to $\tilde{\mathcal{G}}$. However, this happens with negligible probability by a standard birthday bound. Let Q be the number of queries of $\mathcal{A}$ to the random oracle $\mathcal{H}$, the reduction guesses an index $i^* \in \{1, \dots, Q\}$ and simulates the oracle as follows: For $i \neq i^*$, on input x_i the reduction programs the random oracle to

$$\mathcal{H}(x_i) = (g_2^s)^{\rho_i} \text{ where } \rho_i \leftarrow\$ \mathbb{Z}_p.$$

On the other hand, on input x_{i^*} the reduction sets

$$\mathcal{H}(x_{i^*}) = g_2^z.$$

For the VRF evaluation oracle, we assume without loss of generality that the adversary already queried the input x_i to the random oracle. The reduction proceeds as follows: If $x_i = x_{i^*}$, the reduction aborts and returns a random bit. Otherwise, the reduction computes

$$\pi_i = \left(g_2^{ks}\right)^{\rho_i} \text{ and } y_i = e(g_1, \pi)$$

by fetching the appropriate value of ρ_i. The challenge query of the (algebraic) adversary is interpreted by the reduction as

$$(x^*, \mathsf{vk}_1, \dots, \mathsf{vk}_n) \text{ and } \left\{P_i \text{ such that } g_1^{P_i(1,s,ks,z)} = \mathsf{vk}_i\right\}_{i=1\dots n}$$

where P_i is a multilinear polynomial in the variables $(1, s, ks, z)$, in its coefficient embedding. For notational convenience, we group the coefficients corresponding to the variables $\rho_1, \dots, \rho_Q$ (which are also formal variables from the point of view of the adversary) in the constant term. This is always possible since the reduction samples ρ_i itself and therefore knows the corresponding integer in the plain. The reduction checks if $x^* = x_{i^*}$ and returns a random bit if this is not the case. Else it defines P_0 as

$$P_0 = (0, 0, 1, 0)$$

and computes

$$y^* = \prod_{i=0}^{n} e(g_1, g_2)^{r_i P_i(z/s, z, \{kz, r\}, z^2/s)}.$$

where $r_0, \dots, r_n$ is the output of the random oracle $\tilde{\mathcal{H}} \circ \tilde{\mathcal{G}}$ on input $(x^*, \mathsf{vk}, \mathsf{vk}_1, \dots, \mathsf{vk}_n)$. Finally, the reduction returns whatever bit $\mathcal{A}$ returns.

Note that, in case the reduction guesses correctly the query to the random oracle that contains the challenge input x^*, the answers to the oracle queries of the adversary are perfectly simulated by the reduction. This follows using the same argument as in Thm. 4.1. Furthermore, observe that the reduction is efficient, since it only performs basic group operations. In particular, the reduction is supplied by the challenger all group elements needed to compute the evaluation at the challenge point y^*.

In case where the challenge tuple contains the element $e(g_1, g_2)^{kz}$, we argue that the reduction perfectly simulates the view of the adversary with the bit $b = 0$. To see why this is the case, it suffices to observe that

$$\begin{aligned}
y^* &= \prod_{i=0}^{n} e(g_1, g_2)^{r_i P_i(z/s, z, kz, z^2/s)} \\
&= e(g_1, g_2)^{r_0 kz} \prod_{i=1}^{n} e(g_1, g_2)^{r_i P_i(z/s, z, kz, z^2/s)} \\
&= e(g_1, \mathcal{H}(x^*))^{r_0 k} \prod_{i=1}^{n} e(g_1, \mathcal{H}(x^*))^{r_i P_i(1/s, 1, k, z/s)} \\
&= e(g_1, \mathcal{H}(x^*))^{r_0 k} \prod_{i=1}^{n} e(g_1, \mathcal{H}(x^*))^{r_i \mathsf{DLog}_S(\mathsf{vk}_i)} \\
&= e(g_1, \mathcal{H}(x^*))^{r_0 k} \prod_{i=1}^{n} e(g_1, \mathcal{H}(x^*))^{r_i \mathsf{VKey}^{-1}(\mathsf{vk}_i)} \\
&= \mathsf{Agg}\left(x^*, \mathsf{vk}, e(g_1, \mathcal{H}(x^*))^{k}, \left\{\mathsf{vk}_i, e(g_1, \mathcal{H}(x^*))^{k_i}\right\}_{i=1\ldots n}\right)
\end{aligned}$$

which is precisely what the attacker is expecting. On the other hand, in case the challenge tuple contains the element $e(g_1, g_2)^r$, we argue that the output y^* computed by the reduction is uniform, and therefore perfectly simulates the case where $b = 1$, except with negligible probability. Let

$$\tilde{P} = \sum_{i=0}^{n} r_i P_i,$$

then, since $\mathbb{Z}_p$ is a field, the y^* computed by the reduction is uniform if and only if the third coefficient of $\tilde{P}$ is non-zero. Since the tuple $r_0, \ldots, r_n$ is sampled uniformly by the reduction *after* the polynomials $P_0, \ldots, P_n$ are fixed, by Lmm. 4.3, this happens with probability at most $1/p$, which is negligible.

The above argument shows that the reduction succeeds perfectly in case of a correct guess and otherwise returns a random bit. Since this happens with probability at least $1/Q$, the success probability of the reduction in distinguishing the two distributions is at most a polynomial factor smaller than to that of $\mathcal{A}$. This contradicts the ABDH assumption and concludes our proof. □

Aggregate Uniqueness. It can be easily shown that the construction satisfies aggregate uniqueness, unconditionally.

Theorem 4.5. The construction as described above satisfies aggregate uniqueness.

Proof. The proof follows immediately by observing that the aggregate verification key

$$\tilde{\mathsf{vk}} = \prod_{i=1}^{n} \mathsf{vk}_i^{r_i} = S^{\sum_{i=1}^{n} k_i \cdot r_i} = S^{\tilde{k}}$$

is a well-formed VRF key, and thus aggregate uniqueness follows from the uniqueness of the base construction. □

Aggregate Binding. In the following we show that the scheme satisfies aggregate binding. In fact, we show a slightly stronger statement, namely that aggregate binding holds against *unbounded* adversaries that make a polynomial number of queries to the random oracle.

Theorem 4.6. The construction as described above satisfies aggregate binding in the random oracle model.

Proof. Given the aggregate secret key $\tilde{\mathsf{vk}}$, the image-proof pair is unique, thus the only way to break the aggregate uniqueness is to find a collision in the images. That is, for any given set of verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$, the adversary must find a point x^* and two tuples $(y_1, \ldots, y_n) \neq (y_1^*, \ldots, y_n^*)$ such that

$$\mathsf{Agg}\left(x^*, \{\mathsf{vk}_i, y_i\}_{i=1\ldots n}\right) = \mathsf{Agg}\left(x^*, \{\mathsf{vk}_i, y_i^*\}_{i=1\ldots n}\right).$$

In the following we argue that this happens only with negligible probability, over the random choice of $\tilde{\mathcal{G}}$ and $\tilde{\mathcal{H}}$. Recall that the aggregation algorithm computes

$$(r_1, \ldots, r_n) \leftarrow \tilde{\mathcal{H}}\left(\tilde{\mathcal{G}}(x, \mathsf{vk}_1, \ldots, \mathsf{vk}_n, y_1, \ldots, y_n)\right).$$

and outputs $\tilde{y} = \prod_{i=1}^{n} y_i^{r_i}$. First, note that we can equivalently analyze the experiment where $\tilde{\mathcal{H}} \circ \tilde{\mathcal{G}}$ is simulated by lazy sampling (see the proof of Thm. 4.4), with only a negligible loss in the advantage. Second, observe that, except for the tuple $(y_1, \ldots, y_n) = (1, \ldots, 1)$, the output of the aggregation algorithm is uniformly distributed in $\mathbb{G}_1$. It follows that, for any (possibly unbounded) algorithm making a polynomial number of queries to the random oracle, the probability of finding a collision is negligible (in λ) by a standard birthday bound. □

4.3 Anonymity

The notion of anonymity, as defined in [GOT19] requires that the verification key of the VRF can be updated to a new verification key

$$\mathsf{vk}' \leftarrow \mathsf{Update}(\mathsf{vk}, k; \rho)$$

where ρ denotes the random coins used by the Update algorithm, in such a way that the updated key vk' is not linkable to the original key vk. Furthermore, there exists an updated proof algorithm UProve that, on input the secret key k and the random coins ρ used during the update procedure, produces a valid proof π' to correctly verify the image of the VRF at any point. We define more formally the anonymity property below. Note that, compared with the syntax of [GOT19], we define a more permissive variant, where the update algorithm is allowed to take as input the secret key as well. Nevertheless, this variant still suffices for all applications presented in [GOT19].

Definition 4.7 (Anonymity)**.** A VRF satisfies *anonymity* if there exists a negligible function μ such that for $\lambda \in \mathbb{N}$ and all admissible PPT adversaries $\mathcal{A}$ it holds that

$$\left|\frac{1}{2} - \Pr\left[1 \leftarrow \mathsf{ExpAnon}_{\mathcal{A}}(1^\lambda)\right]\right| = \mu(\lambda)$$

where the experiment $\mathsf{ExpAnon}_{\mathcal{A}}$ is defined as follows.

- The challenger samples a reference string $\mathsf{crs} \leftarrow\$ \mathsf{Setup}(1^\lambda)$ and two key pairs

  $$(k_0, \mathsf{vk}_0) \leftarrow\$ \mathsf{Gen}(\mathsf{crs}) \text{ and } (k_1, \mathsf{vk}_1) \leftarrow\$ \mathsf{Gen}(\mathsf{crs})$$

  and sends $(\mathsf{crs}, \mathsf{vk}_0, \mathsf{vk}_1)$ to $\mathcal{A}$.
- $\mathcal{A}$ outputs a challenge input x^*.
- The challenger flips a coin $b \leftarrow\$ \{0,1\}$ and updates the verification key $\mathsf{vk}^* \leftarrow \mathsf{Update}(\mathsf{vk}_b, k_b)$. Denote by ρ the random coins of the Update procedure, the challenger computes

  $$y^* \leftarrow \mathsf{Eval}(k_b, x^*) \quad \text{and} \quad \pi^* \leftarrow \mathsf{UProve}(k_b, \rho, x^*)$$

  and sends $(\mathsf{vk}^*, y^*, \pi^*)$ to $\mathcal{A}$.
- In the end of the experiment, $\mathcal{A}$ outputs a guess b^*. The experiment returns 1 if and only if $b^* = b$.

Update Algorithm. Next, we describe an update algorithm suitable for our construction. We stress here that the updated key belongs to a different domain than the original key, and in particular it no longer satisfies the same homomorphic properties. Nevertheless, this property can still be useful in settings where one performs the homomorphic computation *before* updating the key.

On input a verification key $\mathsf{vk} = S^k = g_1^{ks}$, the update algorithm Update samples a uniform $\kappa \leftarrow\$ \mathbb{Z}_p$ and computes the updated key vk' as

$$\mathsf{vk}' = \left(V = g_1^{s\kappa}, T = g_1^{k/\kappa}\right).$$

The updated proof of correct evaluation π' at point x is computed as $\pi' = \mathcal{H}(x)^\kappa$ and is verified by checking the equations

$$y \stackrel{?}{=} e\left(T, \pi'\right) \text{ and } e\left(S, \pi'\right) \stackrel{?}{=} e\left(V, \mathcal{H}(x)\right).$$

Both equations are satisfied with probability 1 since

$$e\left(T, \pi'\right) = e\left(g_1^{k/\kappa}, \mathcal{H}(x)^\kappa\right) = e(g_1, \mathcal{H}(x))^k = y$$

and

$$e\left(V, \mathcal{H}(x)\right) = e\left(g_1, \mathcal{H}(x)\right)^{s\kappa} = e\left(S, \pi'\right).$$

Furthermore, for any given V and T, the value of π' is uniquely determined.

Analysis. The following theorem shows that the update algorithm as defined above satisfies anonymity.

Theorem 4.8. If the group generator $\mathsf{GroupGen}$ is XIBDH-hard, then the construction as described above satisfies anonymity in the random oracle model.

Proof. Observe that the view of the adversary in the experiment with the challenge bit fixed to $b = 0$ consists of the tuple

$$\left(g_1^s, g_1^{sk_0}, g_1^{sk_1}, g_1^{s\kappa}, g_1^{k_0/\kappa}, g_2^z, g_2^{z\kappa}, e(g_1, g_2^{z\kappa})\right)$$

where $g_2^{\tilde{z}} = \mathcal{H}(x^*)$. By the XIBDH Assumption, this distribution is computationally indistinguishable from

$$\left(g_1^s, g_1^{sk_0}, g_1^{sk_1}, g_1^{s\kappa}, g_1^{\delta}, g_2^{z}, g_2^{z\kappa}, e(g_1, g_2^{z\kappa})\right)$$

where $\delta \leftarrow\$ \mathbb{Z}_p$. Another invocation of the XIBDH Assumption allows us to argue that this is computationally indistinguishable from

$$\left(g_1^s, g_1^{sk_0}, g_1^{sk_1}, g_1^{s\kappa}, g_1^{k_1/\kappa}, g_2^{z}, g_2^{z\kappa}, e(g_1, g_2^{z\kappa})\right)$$

which is exactly the view in the experiment with the challenge bit fixed to $b = 1$. □

4.4 Threshold

We briefly outline how our KH-VRF immediately implies a t-out-of-n threshold VRF, exploiting the linearity of the key to combine the construction with a linear secret-sharing scheme. This transformation is analogous to [BLMR13] and therefore we only provide a sketch of the scheme here, and we refer the reader to [BLMR13] for more details. In the threshold variant, we sample a random key $k \leftarrow\$ \mathbb{Z}_p$ and we interpret it as the constant coefficient of a degree $t-1$ univariate polynomial P, where the rest of the coefficients are also sampled uniformly from $\mathbb{Z}_p$. The share of the i-th party consists of the evaluation of the polynomial $k_i = P(i)$ at point i. For any point $x \in \{0,1\}^\lambda$, each party publishes

$$y_i = e(g_1, \mathcal{H}(x))^{k_i} \quad \text{and} \quad \pi_i = \mathcal{H}(x)^{k_i}.$$

Given t such share, it is possible to publicly reconstruct a valid image-proof pair, by running Lagrange interpolation in the exponent, i.e., computing

$$y = \prod_{i=1}^{t} y_i^{L_i(0)} \quad \text{and} \quad \pi = \prod_{i=1}^{t} \pi_i^{L_i(0)}$$

where L_i is the i-th Lagrange polynomial.

5 Aggregate VRFs from LWE

In the following we describe a generic approach to construct a VRF from generic cryptographic primitives, which can be all instantiated from the standard learning with errors (LWE) assumption [Reg05].

5.1 Information Theory

Recall the definition of the min-entropy of a random variable X as

$$H_\infty(X) = -\log\left(\mathsf{max}_x \Pr[X = x]\right).$$

We recall the definition of average conditional min-entropy in the following.

Definition 5.1 (Average Conditional Min-Entropy)**.** Let X be a random-variable supported on a finite set $\mathcal{X}$ and let Z be a (possibly correlated) random variable supported on a finite set $\mathcal{Z}$. The average-conditional min-entropy $\tilde{H}_\infty(X|Z)$ is defined as

$$\tilde{H}_\infty(X|Z) = -\log\left(\mathbb{E}_z\left[\mathsf{max}_{x\in\mathcal{X}}\Pr[X = x|Z = z]\right]\right).$$

It is shown in [DRS04, DORS08] that the average conditional min-entropy satisfies a *chain rule*, that is

$$\tilde{H}_\infty(X|Z) \geq H_\infty(X) - H_0(Z) \tag{1}$$

where $H_0(Z)$ denotes the logarithm of the size of the support of Z. Next, we recall the definition of a seeded randomness extractor.

Definition 5.2 (Extractor)**.** A function $\mathsf{Ext} : \{0,1\}^d \times \mathcal{X} \to \{0,1\}^\ell$ is called a seeded strong average-case (k, ε)-extractor, if it holds for all random variables X with support $\mathcal{X}$ and Z defined on some finite support that if $\tilde{H}_\infty(X|Z) \geq k$, then it holds that the statistical distance of the following distributions is a most ε

$$(\mathsf{seed}, \mathsf{Ext}(\mathsf{seed}, X), Z) \approx_\varepsilon (\mathsf{seed}, U, Z)$$

where $\mathsf{seed} \leftarrow \{0,1\}^d$ and $U \leftarrow \{0,1\}^\ell$.

Recall that a hash function $\mathsf{Hash} : \mathcal{X} \to \mathcal{Y}$ is a universal hash if for all $x \neq x' \in \mathcal{X}$ it holds that

$$\Pr[\mathsf{Hash}(x) = \mathsf{Hash}(x')] \leq \frac{1}{|\mathcal{Y}|}$$

where the probability is taken over the choice of the hash function. It is shown [DRS04, DORS08] that any universal hash function is an average-case randomness extractor.

Lemma 5.3 (Leftover Hash Lemma)**.** Let X be a random-variable supported on a finite set $\mathcal{X}$ and let Z be a (possibly correlated) random variable supported on a finite set $\mathcal{Z}$ such that $\tilde{H}_\infty(X|Z) \geq k$. Let $\mathsf{Hash} : \mathcal{X} \to \{0,1\}^\ell$, where $\ell \leq k - 2\log\left(\frac{1}{\varepsilon}\right)$, be a family of universal functions. Then Hash is a seeded strong average-case (k, ε)-extractor.

5.2 Cryptographic Building Blocks

Before presenting our scheme, we recall some useful cryptographic building blocks.

Statistically-Binding Commitments. A commitment scheme $\mathsf{Com}(m; r)$ allows one to commit to a message m without revealing it. We require the commitment scheme to be statistically binding and for all messages m_0 and m_1 we require that the distributions

$$\mathsf{Com}(m_0; r) \approx \mathsf{Com}(m_1; r) \quad \text{where } r \leftarrow \{0,1\}^\lambda$$

are computationally indistinguishable (computationally hiding). In this work we assume that the verification of the commitment is canonical: Given the message m and the random coins r used in the commitment phase, the verifier simply recomputes the commitment and checks whether it matches the one sent in the commitment phase. In the common reference string model, such commitments can be built from any one-way function [Nao89].

Simulation Extractable Statistically-Sound NIZK. Let $\mathcal{L}$ be an NP-language with relation $\mathcal{R}$. We say that a statement $x \in \mathcal{L}$ if there exists a witness w such that $\mathcal{R}(x, w) = 1$. A non-interactive zero-knowledge (NIZK) proof for NP [FLS90], allows one to produce a proof π that certifies the membership for a given language, that reveals nothing but its validity. A NIZK assumes a common reference string crs and satisfies the following properties:

- Soundness: For all $x \notin \mathcal{L}$, there does not exist an accepting proof π, with overwhelming probability over the random choice of crs.
- Zero-Knowledge: There exists a simulator Sim such that for all x and w such that $\mathcal{R}(x, w) = 1$ it holds that the following distributions are computationally indistinguishable

  $$(\mathsf{crs}, \pi) \approx \mathsf{Sim}(1^\lambda, x)$$

  where crs and π are honestly computed.

In this work we are interested in a stronger property, namely simulation-sound extractability, which requires that there exists an efficient extractor Ext that (given a trapdoor sampled together with the crs) recovers the witness from any valid proof. Such extractor is required to succeed even if an adversary sees arbitrary simulated proofs. NIZK for NP are known from LWE [PS19] and can be generically lifted to the simulation-extractable settings [Sah99].

Somewhere-Extractable BARGs for NP. We informally recall the notion of somewhere-extractable batch-arguments (BARGs) for NP, and we refer the reader to [CJJ21a, KLVW23, BBK+23] for a precise definition. A BARG allows one to prove n instances of a given NP language with proof size and verification time logarithmic in n. More precisely, we say that an instance (M, x, n, T) is in the batched language $\mathcal{L}$, if there exist witnesses $(w_1 \dots w_n)$ such that

$$\left\{\mathsf{M}(i, x, w_i) \stackrel{?}{=} 1\right\}_{i=1\dots n}$$

and furthermore M accepts within T steps. A BARG for NP allows one to compute a proof π for such instances, with the requirement that the size of π is linear in the maximum size of each witness (ignoring polynomial factors in the security paramter) and furthermore the verifier runtime is bounded by a polynomial in the size of the proof, and a linear function in the size of the intance. In particular, note that the verifier runtime is polylogarithmic in n. We require that a BARG satisfies the following properties:

- Index-Hiding: The setup algorithm of a BARG $\mathsf{Setup}(1^\lambda, i)$ takes as additional input an index $i \in \{1 \dots n\}$ and we require that for all i_0 and i_1 the following distributions

  $$\mathsf{Setup}(1^\lambda, i_0) \approx \mathsf{Setup}(1^\lambda, i_1)$$

  are computationally indistinguishable.
- Somewhere Extractability: There exists an efficient extractor Ext such that, for any index $i \in \{1 \dots n\}$, if the setup algorithm is run on input i, then Ext can recover a valid witness w_i from any accepting proof.

For convenience we say that a BARG is extractable on i, if the setup is initialized on index i. Note that the latter property implies the more standard notion of soundness. BARGs satisfying these properties can be built from a variety of assumptions [CJJ21a, CJJ21b, WW22, KLVW23, CGJ$^+$23], including LWE.

Function-Binding Hash with Local Openings. In the following we present an instantiation of a tree-based function-binding hash. The construction extends prior works [FWW23, BBK$^+$23] by generalizing the class of functions supported beyond one-bit predicates, although the main ideas are very similar, tracing back to the work of [HW15]. Our hash function assumes the existence of a rate-1 levelled fully-homomorphic encryption (FHE) scheme [Gen09, BV11], which in turn can be constructed from LWE [BDGM19, GH19]. For notational convenience, we assume that the domain of the FHE ciphertexts and the domain of the leaves of the tree coincide (this is without loss of generality since we can always pad the blocks of the hash to satisfy this requirement).

Before proceeding with the description of the construction, let us establish some notation: We consider a binary tree with n leaves, where each node in the tree is identified by its canonical bitstring (e.g., the left-most leaf is identified by the string $0^{\log(n)}$). Note that the length of the string is equivalent to the depth of the node. We further refer to a node (except for the root of the tree) as either a 0-node or a 1-node, depending on whether they are the left or the right child of their parent node. A root-to-leaf path consists of the $2\log(n) - 1$ nodes of the tree,[1] and can be checked for validity by verifying that the children correctly hash to the parent node. We are now ready to describe the algorithms of the hash function.

- Setup: The setup consists of a uniformly sampled rate-1 FHE public key pk, along with a ciphertext
$$C_{\mathsf{node}} = \mathsf{FHEnc}(\mathsf{pk}, \mathsf{id}^*)$$
which is parsed as an encryption of the identifier id^* of a node in the tree.

- Hash: The hash function Hash is the standard Merkle tree, except that we define a particular underlying two-to-one function. Specifically, we define such a function to compute:
$$C = \mathsf{FHEval}(\mathsf{pk}, f_{\mathsf{node}_0,\mathsf{node}_1,\mathsf{id}_0,\mathsf{id}_1}, (C_{\mathsf{node}}, \mathsf{node}_0, \mathsf{node}_1)).$$
Note that the variables $(\mathsf{node}_0, \mathsf{node}_1)$ are parsed both as bitstrings (when hardwired in the description of the function) and as ciphertexts (when taken as input for the homomorphic evaluation). The function $f_{\mathsf{node}_0,\mathsf{node}_1,\mathsf{id}_0,\mathsf{id}_1}$ takes as input a node identifier id^* and two messages m_0 and m_1 and is defined as follows.
  - If $\mathsf{id}_0 = \mathsf{id}^*$: Return node_0.
  - Else if $\mathsf{id}_1 = \mathsf{id}^*$: Return node_1.
  - Else if id_0 is a prefix of id^*: Return m_0.
  - Else if id_1 is a prefix if id^*: Return m_1.
  - Else: Return 0.

[1] Although half of these nodes can be omitted, we favor this slightly redundant representation for notational convenience.

To see why the construction satisfies the standard notion of compactness (and, consequently, verification efficiency) for a Merkle-tree hash functions, it suffices to upperbound the size of the computed ciphertext at each layer of the tree. Let s_i be the size of the nodes at depth i (they all have the same size). Then the size of the evaluated ciphertext and, consequently, the nodes at the $(i-1)$-th layer is bounded by

$$s_{i-1} = s_i + \mathsf{poly}(\lambda)$$

since the FHE scheme has rate-1 for the evaluated ciphertexts and the maximum size of the output of f is s_i. Setting $s_{\log(n)}$ (the size of the leaves) to be a fixed polynomial in the security parameter, and unrolling the recursion, we obtain that the size of all nodes in the tree is bounded by

$$\log(n) \cdot \mathsf{poly}(\lambda) = \mathsf{poly}(\lambda)$$

by upper bounding $\log(n) \leq \lambda$. Note that this also implies that the size of a root-to-leaf path is also bounded by a polynomial in the security parameter.

Let us now discuss a few properties that we are going to use later in our construction. First of all, it can be verified that the root ciphertext C encodes the node of the tree corresponding to the identifier id^*. This property is analogous to the somewhere statistically binding (SSB) guarantee from [HW15], except that we generalize it to non-leaf nodes. In a slight abuse of notation, we say that Hash is SSB on id^*, if $C_{\mathsf{node}} = \mathsf{FHEnc}(\mathsf{pk}, \mathsf{id}^*)$. By default we set the hash function to be SSB on $0^{\log(n)}$. Finally, a straightforward reduction to the semantic security of the FHE scheme establishes that

$$\mathsf{FHEnc}(\mathsf{pk}, \mathsf{id}_0^*) \approx \mathsf{FHEnc}(\mathsf{pk}, \mathsf{id}_1^*)$$

for all node identifiers id_0^* and id_1^*. In particular, this means that all modes are computationally indistinguishable.

Obfuscation of Compute-and-Compare Branching Programs. We recall the notion of obfuscation for branching programs [WZ17, GKW17]. Given a branching program f, one can compute an obfuscation $\mathsf{Obf}(f)$, which is parametrized by random matrices $\{\mathbf{A}_{i,b}\}$ and that satisfies the following properties:

- For any x where $f(x) = y \in \{0,1\}^\lambda$ it holds that

  $$\mathsf{Obf}(f)(x) = \mathbf{D}_1 \| \dots \| \mathbf{D}_\lambda \quad \text{where} \quad \mathbf{D}_i = \mathbf{S}_x \mathbf{A}_{i,y_i} + \mathbf{E}_i$$

  where the matrix $\mathbf{S}_x$ is an invertible matrix and it is a public function of x and in particular is independent of $\{\mathbf{A}_{i,b}\}$. Furthermore $\|\mathbf{E}_i\|_\infty \leq B$, where B is some a-priori fixed bound.

- There exists a simulator Sim such that

  $$\mathsf{Obf}(f) \approx \mathsf{Sim}(1^\lambda, 1^{|f|}).$$

For convenience, we define the function Round to be the rounding function that discards the least significant bits of a matrix (component-wise). In particular, we require that for a uniformly sampled $\mathbf{A}$ and all $\mathbf{E}$ such that $\|\mathbf{E}\|_\infty \leq \lambda B$ it holds that

$$\mathsf{Round}\left(\mathbf{A} + \mathbf{E}\right) = \mathsf{Round}\left(\mathbf{A}\right)$$

except with negligible probability.

5.3 Base Construction

As the base VRF construction we use the folklore commit-and-prove VRF scheme [MRV99], which we recall in the following. The verification key for the VRF consists of a staistically binding commitment $c = \mathsf{Com}(k; r)$ where k is the key of a pseudorandom function (PRF) [GGM84]. The evaluation algorithm (using the secret key k) simply evaluates the PRF at point x. To prove the correctness of a image y, one simply computes a NIZK for the language:

$$\mathcal{L} = \left\{(c, y) : \exists\ (r, k) \text{ such that } c \overset{?}{=} \mathsf{Com}(k; r) \text{ and } y \overset{?}{=} \mathsf{PRF}_k(x)\right\}.$$

For technical reasons, in our construction we will use a simulation-extractable NIZK [Sah99] and we will include the image y also in the proof of evaluation correctness (which will be checked for consistency). Note that all ingredients needed for this construction can be instantiated from LWE. However, we explicitly point out that this VRF satisfies a slightly weaker notion of uniqueness: While the image y is uniquely determined by vk, the proof π can be randomized.

5.4 Aggregation

Our aggregation uses the following ingredients: The VRF construction as described above, a somewhere extractable BARG for NP, a function-binding hash function Hash, and a compute-and-compare obfuscation. Note that, as discussed above, all of the above building blocks admit construction in the standard model, assuming the intractability of LWE. We augment the setup of the VRF to include the public parameters of these building blocks, along with a simulated circuit $\tilde{C} = \mathsf{Sim}(1^\lambda, 1^{|f|})$, where f is the circuit describing the FHE decryption, which can be expressed as a branching program [BDGM19].

- $\mathsf{Agg}(x, \{\mathsf{vk}_i, y_i\}_{i=1\dots n})$: Compute

  $$\tilde{y} = \mathsf{Round}\left(\sum_{i=1}^{\lambda} \mathbf{D}_i\right) \quad \text{and} \quad \tilde{\mathsf{vk}} = \mathsf{Hash}(\mathsf{vk}_1, \dots, \mathsf{vk}_n)$$

  where $\tilde{C}\left(\mathsf{Hash}(y_1, \dots, y_n)\right) = \mathbf{D}_1 \| \dots \| \mathbf{D}_\lambda$.

- $\mathsf{AggProve}(x, \{\mathsf{vk}_i, \pi_i\}_{i=1\dots n})$: Compute $y' = \mathsf{Hash}(y_1, \dots, y_n)$, where the image y_i can be read off the proof π_i. Then compute two BARGs for NP, denoted by σ_0 and σ_1, certifying that, for all $i \in \{1, \dots, n\}$, there exists a tuple $(y_i, \mathsf{path}_i, \pi_i)$ such that:

  - The path_i is a valid root-to-leaf path for the tree rooted in y' with leaf y_i.
  - The proof π_i is a valid VRF proof for y_i being the correct evaluation of the VRF at point x under key vk_i.

  Return $\tilde{\pi} = (y', \sigma_0, \sigma_1)$.

- $\mathsf{AggVerify}(\{\mathsf{vk}_i\}_{i=1\dots n}, x, \tilde{y}, \tilde{\pi})$: Verify that

  - The BARGs σ_0 and σ_1 correctly verify.
  - $\tilde{y} \overset{?}{=} \mathsf{Round}\left(\sum_{i=1}^{\lambda} \mathbf{D}_i\right)$ where $\tilde{C}\left(y'\right) = \mathbf{D}_1 \| \dots \| \mathbf{D}_\lambda$.

Correctness follows immediately by the correctness of the underlying building blocks, whereas it can be verified that the aggregates are compact by appealing to the efficiency properties of the function-binding hash and the BARGs for NP. Next, we show that our construction satisfies all properties for an aggregate VRF.

Aggregate Pseudorandomness. We show that our construction satisfies aggregate pseudorandomness.

Theorem 5.4. If Hash is function binding and the base VRF is as described above, the construction as described above satisfies aggregate pseudorandomness.

Proof. The proof proceeds by introducing a series of hybrids and arguing that adjacent experiments are indistinguishable.

- Hybrid 0: This is the original experiment as defined in Def. 3.6.
- Hybrid 1: In this hybrid, we run the NIZK extractor on the challenge keys, instead of (inefficiently) inverting the adversarially-chosen keys. Indistinguishability follows by the extractability of the NIZK. Note that in this hybrid the challenger runs in polynomial time.
- Hybrid 2: In this hybrid, we simulate the NIZK of the VRF when answering the oracle queries of the adversary. This switch is computationally indistinguishable by the zero-knowledge of the NIZK.
- Hybrid 3: In this hybrid, we modify the commitment c in the challenge VRF to $c = \mathsf{Com}(0; r)$. Indistinguishability follows from the computational hiding of the commitment scheme.
- Hybrid 4: In this hybrid, we switch the evaluation of the challenge VRF with a truly random function, which can be efficiently simulated via lazy sampling. This hybrid is indistinguishable from the previous one by the pseudorandomness of the VRF.
- Hybrid 5: In this hybrid, we guess a random index $i^* \in \{1, \ldots, n\}$ and we abort if the challenge key is not in the i^* position. Since this happens with probability exactly $1/n$ it follows that, conditioned on abort not happening, the distinguishing advantage of the adversary must still be at least inverse polynomial.
- Hybrid 6: In this hybrid, we make the hash function SSB on the i^* leaf. This step is indistinguishable by the mode indistinguishability of Hash.
- Hybrid 7: In this hybrid, we compute $\tilde{C} = \mathsf{Obf}(\mathsf{FHDec}(\mathsf{sk}, \cdot))$, where sk is the secret key of the FHE scheme. This step is computationally indistinguishable by the security of the compute-and-compare obfuscation.

Note that, in the last hybrid we that, by the correctness of the compute-and-compare obfuscator

$$\tilde{C}(\mathsf{Hash}(y_1, \ldots, y_n)) = \mathbf{D}_1 \| \ldots \| \mathbf{D}_\lambda \quad \text{where} \quad \mathbf{D}_i = \mathbf{S}\mathbf{A}_{i, y_{i^*,i}} + \mathbf{E}_i$$

where we denote by $y_{i^*,i}$ the i-th bit of y_{i^*} which is the challenge image of the VRF. This holds because $\mathsf{Hash}(y_1, \ldots, y_n)$ is an FHE encryption of the i^* leaf, as described in the above hybrid. Recall that the aggregate $\tilde{y}$ is defined as

$$\mathsf{Round}\left(\sum_{i=1}^{\lambda} \mathbf{D}_i\right) = \mathsf{Round}\left(\mathbf{S}^* \sum_{i=1}^{\lambda} \mathbf{A}_{i, y_{i^*,i}} + \mathbf{E}_i\right) = \mathsf{Round}\left(\mathbf{S}^* \sum_{i=1}^{\lambda} \mathbf{A}_{i, y_{i^*,i}}\right)$$

where $\mathbf{S}^*$ is independent of $\{\mathbf{A}_{i,b}\}$. By Eq. 1 (Chain Rule), conditioned on $\mathbf{S}^*$, the string y_{i^*} has at least $\lambda - |\mathbf{S}^*|$ bits of min-entropy. For a large enough λ, by Lmm. 5.3, it holds that

$$\left(\mathbf{S}^*, \sum_{i=1}^{\lambda} \mathbf{A}_{i,y_{i^*,i}}\right) \approx (\mathbf{S}^*, \mathbf{U})$$

are statistically close, where $\mathbf{U}$ is uniformly sampled. Since $\mathbf{S}^*$ is invertible with all but negligible probability, it follows that $\mathbf{S}^*\mathbf{U}$ is also a uniformly sampled matrix. Consequently, so it is its rounding.

□

Aggregate Uniqueness. In the following, we prove that our construction satisfies aggregate uniqueness, although only in a computational sense. I.e., we show that there exists no efficient attacker that, given an honestly-computed aggregate key $\tilde{\mathsf{vk}}$ and image $\tilde{y}$, can produce a valid proof for a different aggregate image.

Theorem 5.5. If Hash is function binding and mode indistinguishable, the BARG is somewhere extractable and index-hiding, and the base VRF is as described above, the construction as described above satisfies aggregate uniqueness.

Proof. Let us assume towards contradiction that there exists an adversary such that, given an honestly computed aggregate

$$(\tilde{y}, \tilde{\mathsf{vk}}) \leftarrow \mathsf{Agg}(x, \{\mathsf{vk}_i, y_i\}_{i=1\ldots n})$$

on valid images y_i, returns an image $y^* \neq \tilde{y}$ and a proof π^* that certifies that y^* is a valid aggregate under $\tilde{\mathsf{vk}}$, with at least inverse-polynomial probability. Note that the proof π^* consists of

- A string y^{**} such that y^*=$\mathsf{Round}\left(\sum_{i=1}^{\lambda} \mathbf{D}_i\right)$ where $\tilde{C}\left(y^{**}\right) = \mathbf{D}_1 \| \ldots \| \mathbf{D}_\lambda$.
- Two BARGs σ_0^* and σ_1^* that certify the validity of y^*.

Note that, since $\tilde{C}$ is a deterministic function, it must be the case that $y^{**} \neq y'$, where $y' = \mathsf{Hash}(y_1, \ldots, y_n)$ is the honestly computed hash. By the extractability of the BARGs, we can efficiently recover from σ_0^* and σ_1^* two root-to-leaf paths path_0^* and path_1^* corresponding to the leaf indexed by $0^{\log(n)}$. We can also efficiently compute the same root-to-leaf path corresponding to the honest computation of $\mathsf{Hash}(y_1, \ldots, y_n)$, which we denote by path. Next, we compare path_0^* with path and consider three cases:

1. The two root-to-leaf paths path_0^* and path are identical: This cannot happen, since otherwise the roots would also be identical, which we already established that is not the case.
2. The two root-to-leaf paths path_0^* and path differ in the leaf node: To see why this case happens only with negligible probability, it suffices to recall that the extractor of the BARG is guaranteed to extract valid witnesses for the language, which in particular means that the extracted y_1^* (the leaf contained in the extracted path) is a valid VRF image and π_1^* (the extracted proof) is an accepting proof. However, this contradicts the uniqueness of the VRF since, by assumption, $y_1^* \neq y_1$.

3. The two root-to-leaf paths path_0^* and path differ in a non-leaf node: In this case, the extracted paths differ from the honest one at some 1-node at depth $i \in \{1, \ldots, \log(n) - 1\}$. Bounding this case is somewhat more involved and we do this below.

It is easy to see that the three cases above describe the entire probability space, so what is left to be shown is to bound the probability of the last event to happen. In the following, we show that this probability is indeed negligible. Let $\mathsf{DIFF}(d)$ denote the event that the extracted paths differ at the 1-node at the d-th layer, and let i be the largest such index. We can now rephrase the above condition as the existence of an index $i \in \{1, \ldots, \log(n) - 1\}$ such that

$$\Pr\left[\mathsf{DIFF}(i) \;\middle|\; \mathsf{Setup}_0(1^\lambda, 1) \text{ and } \mathsf{Setup}_1(1^\lambda, 1) \text{ and } \mathsf{Hash} \text{ is SSB at } 0^{\log(n)}\right] \geq \frac{1}{\mathsf{poly}(\lambda)}$$

where Setup_0 and Setup_1 are the setup algorithms of the two BARGs. Denote by $j \in \{1 \ldots, n\}$ the leaf corresponding to the identifier

$$\mathsf{id} = 0 \ldots 0 \underbrace{1}_{i\text{-th bit}} 0 \ldots 0$$

and let id_i be the same string but truncated after the 1. We claim that also

$$\begin{aligned}&\Pr\left[\mathsf{DIFF}(i) \;\middle|\; \mathsf{Setup}_0(1^\lambda, 1) \text{ and } \mathsf{Setup}_1(1^\lambda, 1) \text{ and } \mathsf{Hash} \text{ is SSB at } 0^{\log(n)}\right]\\ &\approx \Pr\left[\mathsf{DIFF}(i) \;\middle|\; \mathsf{Setup}_0(1^\lambda, 1) \text{ and } \mathsf{Setup}_1(1^\lambda, 1) \text{ and } \mathsf{Hash} \text{ is SSB at } \mathsf{id}_i\right]\end{aligned}$$

by the mode indistinguishability of the SSB hash function, since such an event is efficiently detectable and can be used to distinguish the two modes of the hash function. Next we claim that, similarly, it holds that

$$\begin{aligned}&\Pr\left[\mathsf{DIFF}(i) \;\middle|\; \mathsf{Setup}_0(1^\lambda, 1) \text{ and } \mathsf{Setup}_1(1^\lambda, 1) \text{ and } \mathsf{Hash} \text{ is SSB at } \mathsf{id}_i\right]\\ &\approx \Pr\left[\mathsf{DIFF}(i) \;\middle|\; \mathsf{Setup}_0(1^\lambda, 1) \text{ and } \mathsf{Setup}_1(1^\lambda, j) \text{ and } \mathsf{Hash} \text{ is SSB at } \mathsf{id}_i\right]\end{aligned}$$

by the index-hiding of the BARG, since once again the event is efficiently observable without extracting the BARG σ_0^*. Let us now define the event $\mathsf{DIFF}_1(i)$ as $\mathsf{DIFF}(i)$, except that we look at the extracted path path_1^*, instead of path_0^*. We claim that the probabilities

$$\begin{aligned}&\Pr\left[\mathsf{DIFF}(i) \;\middle|\; \mathsf{Setup}_0(1^\lambda, 1) \text{ and } \mathsf{Setup}_1(1^\lambda, j) \text{ and } \mathsf{Hash} \text{ is SSB at } \mathsf{id}_i\right]\\ &\approx \Pr\left[\mathsf{DIFF}_1(i) \;\middle|\; \mathsf{Setup}_0(1^\lambda, 1) \text{ and } \mathsf{Setup}_1(1^\lambda, j) \text{ and } \mathsf{Hash} \text{ is SSB at } \mathsf{id}_i\right]\end{aligned}$$

are identical, except for negligible factors. This holds because, by definition, node identified by id_i belongs to both root-to-leaf path for the leaves 1 and j. Furthermore, the hash function is statistically bound for the node indexed by id_1. We can now appeal to the index-hiding of the BARG to show that

$$\begin{aligned}&\Pr\left[\mathsf{DIFF}_1(i) \;\middle|\; \mathsf{Setup}_0(1^\lambda, 1) \text{ and } \mathsf{Setup}_1(1^\lambda, j) \text{ and } \mathsf{Hash} \text{ is SSB at } \mathsf{id}_i\right]\\ &\approx \Pr\left[\mathsf{DIFF}_1(i) \;\middle|\; \mathsf{Setup}_0(1^\lambda, j) \text{ and } \mathsf{Setup}_1(1^\lambda, j) \text{ and } \mathsf{Hash} \text{ is SSB at } \mathsf{id}_i\right]\\ &\approx \Pr\left[\mathsf{DIFF}(i) \;\middle|\; \mathsf{Setup}_0(1^\lambda, j) \text{ and } \mathsf{Setup}_1(1^\lambda, j) \text{ and } \mathsf{Hash} \text{ is SSB at } \mathsf{id}_i\right]\end{aligned}$$

where the second equality follows using a similar argument as above. At this point it is useful to recall that the i-th bit of id_1 is 1. In particular, this means that the children of such node must be part of the root-to-leaf path corresponding to j and are therefore part of path_0^*. Note that at least one of these children must differ from the same node in the honest path, since Hash is a deterministic procedure. Let us consider two cases:

- The differing child is a 0-node: In this case, the children of such node are also included in the root-to-leaf path. If this is the case, we simply look at one layer below and check one again which children differ.
- The differing child is a 1-node: Let $i^* > i$ be the depth of such node. In this case the index i is no longer the largest index such that $\mathsf{DIFF}(i)$ holds.

Overall, we have established that there must exists an $i^* > i$ such that

$$\Pr\left[\mathsf{DIFF}(i^*) \;\middle|\; \mathsf{Setup}_0(1^\lambda, j) \text{ and } \mathsf{Setup}_1(1^\lambda, j) \text{ and } \mathsf{Hash} \text{ is SSB at } \mathsf{id}_i\right] \geq \frac{1}{\mathsf{poly}(\lambda)} - \mathsf{negl}(\lambda).$$

Now we can recursively apply the same argument as above, until $i^* = \log(n)$, in which case the differing node is a leaf. We now have reached the case where the differing node is a leaf, which we already argued that contradicts the uniqueness of the VRF. □

Aggregate Binding. Finally, we show that the scheme satisfies aggregate binding.

Theorem 5.6. If Hash is function binding, the BARG is somewhere extractable, and the base VRF is as described above, the construction as described above satisfies aggregate binding.

Proof. The claim follows by a reduction to the SSB of Hash and to the index-hiding of the BARG. The reduction samples a random index $i \in \{1 \dots n\}$ and makes the function Hash SSB on the i-th leaf and the BARGs extractable on index i; If the tuples $(y_1, \dots, y_n)$ and $(y_1^* \dots y_n^*)$ are identical in the i-th element, the reduction aborts. Note that the probability of the latter event happening can only change by a negligible amount, since otherwise one would violate the mode indistinguishability of Hash or the index-hiding of the BARGs. Thus, with at least inverse-polynomial probability, we have that the Hash is SSB at position i and $(y_1, \dots, y_n)$ and $(y_1^* \dots y_n^*)$ differ in the i-th element. This contradicts the verifiability of VRF: Recall that $y_i^* \neq y_i$, which in particular means that y_i^* is not a valid VRF image. However, one can efficiently extract a valid proof for y_i^* from the BARGs, contradicting the uniqueness of the VRF. □

6 Applications

We discuss several motivating applications for the notion of key-homomorphic and aggregate VRFs. In favor of a more intuitive explanation, we keep the following discussion at a high-level and we leave the precise formalization of the security properties and the protocol description as ground for future work. We stress that the purpose of this discussion is to demonstrate the usefulness and the versatility of the notion of our VRFs, but we do not claim that our solutions are superior in *all aspects* compared to existing protocols.

6.1 Distributed VRF without a Trusted Dealer

As a warm-up application, we show how an aggregate VRF imply a simple and efficient construction of an n-out-of-n VRF where the key is distributed across n parties. The key generation is completely non-interactive and performed locally by all parties, which do not be even be aware of each other's existence. In particular, no trusted dealer is needed to distribute the shares of the key. We outline the protocol below.

- *Local Key Generation:* On input the common reference string crs, each party simply samples a key-pair locally

  $$(k, \mathsf{vk}) \leftarrow\$ \ \mathsf{Gen}(\mathsf{crs})$$

  and outputs vk.

- *Shared Key Computation:* The shared key pair is defined to be

  $$\tilde{k} = \sum_{i=1}^{n} k_i \quad \text{and} \quad \tilde{\mathsf{vk}} = \prod_{i=1}^{n} \mathsf{vk}_i.$$

It is easy to see that the resulting key is well-formed by the homomorphic property of the underlying VRF, and that each party can compute the share of the the evaluation at any point x, once again by appealing to the linear homomorphism of the key. The above scheme only ensures correctness against semi-honest parties, whereas if one wants to defend against rogue-key attacks the aggregation function presented in Sec. 4.2 can be used.

6.2 Proof of Stake Blockchains

To see how aggregate VRF provide substantial advantages when applied to proof of stake (PoS) protocols. The discussion here is deliberately informal, and we refer the reader to [FHASW23] for a thorough treatment of aggregatable PoS protocols. Recall that in a PoS protocol [DGKR18] any user in the system uploads their VRF verification key to the blockchain. In each epoch, each user can evaluate the VRF on the slot number (which will play the role as a unique identifier for the epoch) to compute the corresponding image. If the image is less than (a function of) their wealth, then that user is identified as the designated party to generate the new block. Such party can then compute a proof to convince the all other parties that the VRF was evaluated correctly. This way, it can be publicly verified that the user was indeed the winner of the round. In committee-based PoS protocols, such as Algorand [GHM$^+$17], a similar process is performed in order to elect members of a committee, that collectively determine the value of the next block.

Plugging in a KH-VRF in the above protocol, we immediately obtain the following properties:

1. In committee-based PoS, we can aggregate individual proofs

   $$\tilde{\pi} \leftarrow \mathsf{AggProve}(x, \{\mathsf{vk}_i, \pi_i\}_{i=1\ldots n})$$

   so that the size of the proof becomes *independent* of the size of the committee. By the aggregate binding of the VRF, an attacker cannot later change the images in such a way that they will pass the aggregate verification test.

2. One does not need to store a proof for each block (which is needed to make the blockchain publicly verifiable) but one can instead store a *single proof* throughout the lifetime of the system, by recursively aggregating all of the existing proofs. One can still verify that the images are still valid, and an attacker cannot change images after the fact, in such a way that they will pass the aggregate verification test (assuming that the VRF satisfies aggregate binding).

6.3 Verifiable Symmetric-Key Proxy Re-Encryption

The work of [BLMR13] introduces the notion of key-homomorphic pseudorandom functions (KH-PRF), which is analogous to KH-VRF, with the crucial difference that there is no verification procedure for the output of the PRF. In [BLMR13] the authors show a number of applications for this primitive, such as distributed PRFs, symmetric-key proxy re-encryption, updatable encryption, and PRFs secure against related-key attacks. Our key-homomorphic VRF construction can be used as a drop-in replacement to augment all of these applications with an efficient verification procedure.

To exemplify this claim, we discuss the case of symmetric-key proxy re-encryption. Suppose we have a dataset where each message $M_i \in \mathbb{G}_T$ is encrypted as

$$\mathsf{Enc}(k, M_i) = \mathsf{Eval}(k, n_i) \cdot M_i$$

where n_i is a public nonce associated with the i-th location. A trivial solution to update the key k is to download the entire dataset and re-encrypt all data. A more efficient solution is for the client to provide the server with the key $\delta - k$ (where $\delta \in \mathbb{Z}_p$ is the newly sampled key). Then the server can update all ciphertext by computing

$$\mathsf{Enc}(k, M_i) \cdot \mathsf{Eval}(\delta - k, n_i) = \mathsf{Eval}(\delta, n_i) \cdot M_i = \mathsf{Enc}(\delta, M_i).$$

Provided that the server deletes $\delta - k$, it is shown that this protocol satisfies security. If we plug a key-homomorphic VRF in this template, we enable an additional *verification* property: For each ciphertext, the server can efficiently compute a proof that certifies that the update was done correctly. An external auditor, can be convinced that cipheretexts have been correctly updated, without the need to know the secret key $\delta - k$.

Acknowledgements

Work supported by the European Research Council through an ERC Starting Grant (Grant agreement No. 101077455, ObfusQation).

The author wishes to thank the anonymous reviewers of TCC 2024 for their comments.

References

[ASM07] Man Ho Au, Willy Susilo, and Yi Mu. Practical compact e-cash. In Josef Pieprzyk, Hossein Ghodosi, and Ed Dawson, editors, *Information Security and Privacy, 12th Australasian Conference, ACISP 2007, Townsville, Australia, July 2-4, 2007, Proceedings*, volume 4586 of *Lecture Notes in Computer Science*, pages 431–445. Springer, 2007.

[BBK$^+$23] Zvika Brakerski, Maya Farber Brodsky, Yael Tauman Kalai, Alex Lombardi, and Omer Paneth. Snargs for monotone policy batch NP. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 252–283. Springer, 2023.

[BCKL09] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. Compact e-cash and simulatable vrfs revisited. In Hovav Shacham and Brent Waters, editors, *Pairing-Based Cryptography - Pairing 2009, Third International Conference, Palo Alto, CA, USA, August 12-14, 2009, Proceedings*, volume 5671 of *Lecture Notes in Computer Science*, pages 114–131. Springer, 2009.

[BDE$^+$22] Maxime Buser, Rafael Dowsley, Muhammed F. Esgin, Shabnam Kasra Kermanshahi, Veronika Kuchta, Joseph K. Liu, Raphaël C.-W. Phan, and Zhenfei Zhang. Post-quantum verifiable random function from symmetric primitives in pos blockchain. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *Computer Security - ESORICS 2022 - 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26-30, 2022, Proceedings, Part I*, volume 13554 of *Lecture Notes in Computer Science*, pages 25–45. Springer, 2022.

[BDGM19] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles. In Dennis Hofheinz and Alon Rosen, editors, *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 407–437. Springer, 2019.

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 435–464. Springer, 2018.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432. Springer, 2003.

[Bit17] Nir Bitansky. Verifiable random functions from non-interactive witness-indistinguishable proofs. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part II*, volume 10678 of *Lecture Notes in Computer Science*, pages 567–594. Springer, 2017.

[BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 410–428. Springer, 2013.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *J. Cryptol.*, 17(4):297–319, 2004.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 97–106. IEEE Computer Society, 2011.

[CGJ$^+$23] Arka Rai Choudhuri, Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Jiaheng Zhang. Correlation intractability and snargs from sub-exponential DDH. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part IV*, volume 14084 of *Lecture Notes in Computer Science*, pages 635–668. Springer, 2023.

[CJJ21a] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 394–423. Springer, 2021.

[CJJ21b] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Snargs for $\mathcal{P}$ from LWE. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 68–79. IEEE, 2021.

[DDKL24] Bernardo David, Rafael Dowsley, Anders Konring, and Mario Larangeira. MUSEN: Aggregatable key-evolving verifiable random functions and applications. Cryptology ePrint Archive, Paper 2024/628, 2024.

[DGKR18] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 66–98. Springer, 2018.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam D. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam D. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken,* *Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 523–540. Springer, 2004.

[DVW20] Yevgeniy Dodis, Vinod Vaikuntanathan, and Daniel Wichs. Extracting randomness from extractor-dependent sources. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 313–342. Springer, 2020.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *Public Key Cryptography - PKC 2005, 8th International Workshop on Theory and Practice in Public Key Cryptography, Les Diablerets, Switzerland, January 23-26, 2005, Proceedings*, volume 3386 of *Lecture Notes in Computer Science*, pages 416–431. Springer, 2005.

[EKS$^+$21] Muhammed F. Esgin, Veronika Kuchta, Amin Sakzad, Ron Steinfeld, Zhenfei Zhang, Shifeng Sun, and Shumo Chu. Practical post-quantum few-time verifiable random function with applications to algorand. In Nikita Borisov and Claudia Díaz, editors, *Financial Cryptography and Data Security - 25th International Conference, FC 2021, Virtual Event, March 1-5, 2021, Revised Selected Papers, Part II*, volume 12675 of *Lecture Notes in Computer Science*, pages 560–578. Springer, 2021.

[ESLR23] Muhammed F. Esgin, Ron Steinfeld, Dongxi Liu, and Sushmita Ruj. Efficient hybrid exact/relaxed lattice proofs and applications to rounding and vrfs. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part V*, volume 14085 of *Lecture Notes in Computer Science*, pages 484–517. Springer, 2023.

[FHASW23] Nils Fleischhacker, Mathias Hall-Andersen, Mark Simkin, and Benedikt Wagner. Jackpot: Non-interactive aggregatable lotteries. Cryptology ePrint Archive, Paper 2023/1570, 2023.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 33–62. Springer, 2018.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume I*, pages 308–317. IEEE Computer Society, 1990.

[FWW23] Cody Freitag, Brent Waters, and David J. Wu. How to use (plain) witness encryption: Registered abe, flexible broadcast, and more. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August*

20-24, 2023, Proceedings, Part IV, volume 14084 of *Lecture Notes in Computer Science*, pages 498–531. Springer, 2023.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178. ACM, 2009.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th Annual Symposium on Foundations of Computer Science, West Palm Beach, Florida, USA, 24-26 October 1984*, pages 464–479. IEEE Computer Society, 1984.

[GH19] Craig Gentry and Shai Halevi. Compressible FHE with applications to PIR. In Dennis Hofheinz and Alon Rosen, editors, *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 438–464. Springer, 2019.

[GHKW17] Rishab Goyal, Susan Hohenberger, Venkata Koppula, and Brent Waters. A generic approach to constructing and proving verifiable random functions. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part II*, volume 10678 of *Lecture Notes in Computer Science*, pages 537–566. Springer, 2017.

[GHM+17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 51–68. ACM, 2017.

[GKW17] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 612–621. IEEE Computer Society, 2017.

[GLOW21] David Galindo, Jia Liu, Mihai Ordean, and Jin-Mann Wong. Fully distributed verifiable random functions and their application to decentralised random beacons. In *IEEE European Symposium on Security and Privacy, EuroS&P 2021, Vienna, Austria, September 6-10, 2021*, pages 88–102. IEEE, 2021.

[GNP+15] Sharon Goldberg, Moni Naor, Dimitrios Papadopoulos, Leonid Reyzin, Sachin Vasant, and Asaf Ziv. NSEC5: provably preventing DNSSEC zone enumeration. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015.* The Internet Society, 2015.

[GOT19] Chaya Ganesh, Claudio Orlandi, and Daniel Tschudi. Proof-of-stake protocols for privacy-aware blockchains. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 690–719. Springer, 2019.

[HJ16] Dennis Hofheinz and Tibor Jager. Verifiable random functions from standard assumptions. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part I*, volume 9562 of *Lecture Notes in Computer Science*, pages 336–362. Springer, 2016.

[HMW18] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system, 2018.

[HW10] Susan Hohenberger and Brent Waters. Constructing verifiable random functions with large input spaces. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 656–672. Springer, 2010.

[HW15] Pavel Hubácek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 163–172. ACM, 2015.

[Jag15] Tibor Jager. Verifiable random functions from weaker assumptions. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 121–143. Springer, 2015.

[JS04] Stanislaw Jarecki and Vitaly Shmatikov. Handcuffing big brother: an abuse-resilient transaction escrow scheme. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 590–608. Springer, 2004.

[KLVW23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and RAM delegation. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 1545–1552. ACM, 2023.

[KM13] Veronika Kuchta and Mark Manulis. Unique aggregate signatures with applications to distributed verifiable random functions. In Michel Abdalla, Cristina Nita-Rotaru, and Ricardo Dahab, editors, *Cryptology and Network Security - 12th International Conference, CANS 2013, Paraty, Brazil, November 20-22. 2013. Proceedings*, volume 8257 of *Lecture Notes in Computer Science*, pages 251–270. Springer, 2013.

[Koh19] Lisa Kohl. Hunting and gathering - verifiable random functions from standard assumptions with short proofs. In Dongdai Lin and Kazue Sako, editors, *Public-Key Cryptography - PKC 2019 - 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part II*, volume 11443 of *Lecture Notes in Computer Science*, pages 408–437. Springer, 2019.

[LBM20] Bei Liang, Gustavo Banegas, and Aikaterini Mitrokotsa. Statically aggregate verifiable random functions and application to e-lottery. *Cryptogr.*, 4(4):37, 2020.

[Lis05] Moses D. Liskov. Updatable zero-knowledge databases. In Bimal K. Roy, editor, *Advances in Cryptology - ASIACRYPT 2005, 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005, Proceedings*, volume 3788 of *Lecture Notes in Computer Science*, pages 174–198. Springer, 2005.

[Lys02] Anna Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 597–612. Springer, 2002.

[MRV99] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 120–130. IEEE Computer Society, 1999.

[Nao89] Moni Naor. Bit commitment using pseudo-randomness. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 128–136. Springer, 1989.

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 89–114. Springer, 2019.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93. ACM, 2005.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 543–553. IEEE Computer Society, 1999.

[Sco21] Michael Scott. A note on group membership tests for g_1, g_2 and g_t on bls pairing-friendly curves. Cryptology ePrint Archive, Paper 2021/1130, 2021.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer, 1997.

[WW22] Brent Waters and David J. Wu. Batch arguments for sfnp and more from standard bilinear group assumptions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part II*, volume 13508 of *Lecture Notes in Computer Science*, pages 433–463. Springer, 2022.

[WZ17] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under LWE. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 600–611. IEEE Computer Society, 2017.

A Assumptions in the GGM

We briefly discuss why the assumption that we consider are true (unconditionally) in the generic group model (GGM). The following discussion assumes familiarity with basic concepts of the GGM, and we refer the reader to [Sho97] for a thorough introduction. Since both assumptions are static, it suffices to show that there is no way to express that "target" variable τ (i.e., the group element that is different in the two distributions) as a linear combination of other variables in $\mathbb{G}_T$. This is without loss of generality, since we can always move all variables in the target group by computing all possible pairings with the given variables. The claim then follows by Lmm. 4.3.

For Def. 2.2, the adversary's view contain the handles corresponding to the formal variables

$$\left(1, a, b, c, ab, ac, bc, \frac{c}{a}, \frac{c^2}{a}, \tau\right) \text{ where } \tau = \left\{\frac{bc}{a}, r\right\}$$

by taking all possible pairings with the element from the source groups. Since τ is linearly independent from all other variables, our claim follows.

For Def. 2.3, again moving all variables to the target group, we have that the adversary's view contains

$$\left(1, a, ab, ac, d, dc, ad, adc, abd, abdc, acd, adc^2, \tau_0, \tau_1, \tau_2\right)$$

where

$$\tau_0 = \left\{\frac{b}{c}, r\right\} \text{ and } \tau_1 = \left\{\frac{bd}{c}, rd\right\} \text{ and } \tau_2 = \{bd, rdc\}.$$

Once again, the claim follows from the fact that τ_0, τ_1, τ_2 are linearly independent from the rest of the variables.