# Exponent-VRFs and Their Applications

Dan Boneh[1], Iftach Haitner[2,3], and Yehuda Lindell[3]

[1] Stanford University
[2] Tel-Aviv University
[3] Coinbase

**Abstract.** *Verifiable random functions (VRFs)* are pseudorandom functions with the addition that the function owner can prove that a generated output is correct (i.e., generated correctly relative to a committed key). In this paper we introduce the notion of an **exponent-VRF (eVRF)**: a VRF that does not provide its output $y$ explicitly, but instead provides $Y = y \cdot G$ where $G$ is a generator of some finite cyclic group (or $Y = g^y$ in multiplicative notation). We construct eVRFs from DDH and from the Paillier encryption scheme (both in the random-oracle model). We then show that an eVRF can be used to solve several long-standing open problems in threshold cryptography. In particular, we construct (1) a one-round fully simulatable distributed key-generation protocols (after a single two-round initialization phase), (2) a two-round fully simulatable signing protocols for multiparty Schnorr with a deterministic variant, (3) a two-party ECDSA protocol that has a deterministic variant, (4) a threshold Schnorr signing where the parties can later prove that they signed without being able to frame another group, (5) an MPC-friendly and verifiable HD-derivation. Efficient simulatable protocols of this round complexity were not known prior to this work. All of our protocols are concretely efficient.

## 1 Introduction

A *pseudorandom function* (PRF) [23] $F(k, x)$ is a keyed function whose outputs are indistinguishable from random elements in the range of the PRF. In some applications, it is important to force the secret-key owner to always use the same key and to generate correct outputs. A *verifiable random function* (VRF) [30] associates a public verification key $\mathsf{vk}$ with the secret key $k$, and enables the owner to output a proof $\pi$, together with $y = F(k, x)$, that attests to the fact that $y$ is correct.

In this paper, we introduce a VRF enhancement that we call an **exponent VRF (eVRF)**, which is a variant of a VRF that does not provide the VRF's output $y$ explicitly, but rather provides $Y = y \cdot G$ where $G$ is a group generator of some cyclic group $\mathbb{G}$, together with a proof $\pi$ that $Y$ was computed correctly by computing $y = F(k, x)$ and $Y = y \cdot G$. We use the term "exponent VRF", since the VRF output is provided only in the exponent and not in the clear.[4]

eVRFs are useful in settings where the discrete log problem (or DDH) is hard over the group, and a party needs to generate a pseudorandom value $r$, and send $R = r \cdot G$ to other parties. If the sender generates $(r, R)$ using an eVRF, the receiving parties can verify that $R$ is consistent with an initially committed key $k$. Concretely, consider a very basic setting where two parties wish to generate a random group element $R$ in $\mathbb{G}$, where the parties hold shares $r_1$ and $r_2$ such that $(r_1 + r_2) \cdot G = R$, and no party knows $r = r_1 + r_2$. This basic building block is used in distributed key generation, and in ECDSA and Schnorr/EdDSA signing. The naive way of generating $R$ is for each party $P_i$ ($i \in \{1, 2\}$) to choose a random $r_i$ and send $R_i = r_i \cdot G$ to the other party. In such a protocol, however, a cheating $P_2$ can wait to obtain $R_1$ from $P_1$, and can then choose $r$ and compute $R = r \cdot G$ and return $R_2 = R - R_1$ to $P_1$. This enables $P_2$ to single handedly determine the output, while knowing the discrete log $r$ of $R$. This can be mitigated by forcing $P_1$ and $P_2$ to

[4] The use of the term *exponent* comes from multiplicative notation where $g$ is the group generator and $Y = g^y$.

each provide a zero-knowledge proof of knowledge of the discrete log together with their values $R_1$ and $R_2$, respectively. A cheating $P_2$, however, can still receive $R_1$ and then locally try many values $r_2$ and $R_2 = r_2 \cdot G$ until $R = R_1 + R_2$ has some predetermined structure. For example, if $P_2$ wishes the 10 least significant bits of $R$ to equal zero, then it would need to try approximately $2^{10}$ values of $r_2$. In order to prevent such bias, protocols employ a commit-and-open approach: the parties first send *commitments* to their $R_i$ values, and open them in the next round (for simulatability, proofs of knowledge are also included).

An eVRF provides a much simpler construction for this basic building block. Suppose the parties have already generated and shared eVRF public verification keys $\mathsf{vk}_1$ and $\mathsf{vk}_2$. Then they can choose $R_1$ and $R_2$ as the eVRF outputs on some agreed-upon nonce, such as a simple counter. Neither party has any freedom in choosing its value $R_i$. This means that the first naive protocol described above becomes fully secure. In particular, each party sends $R_i$ together with a proof that $R_i$ is the output of its eVRF. The parties then set $R = R_1 + R_2$. Thus, they can generate $R$ with a single message and using only one round, and no party can bias the output in any way, since they are already fully committed to their VRF value. Thus, an eVRF eliminates the need for commit-and-open and enables us to save a full round of communication in threshold signing and key generation protocols.

**Applications.** Using an eVRF, we are able to provide several new results in threshold signing.

1. We construct the first concretely-efficient, two-round, *fully simulatable* multiparty Schnorr signing protocol. Previous two-round protocols are either proven via a game-based definition (e.g., [24]) or are not concretely efficient (e.g., [19]).
2. We construct the first concretely-efficient, two-round, two-party ECDSA signing protocol with full simulatability. Previous work achieving two-round two-party ECDSA signing did not use a standard signing functionality [16].
3. We construct the first concretely-efficient, two-round, deterministic signing protocols for multiparty Schnorr and two-party ECDSA. Previous protocols use garbled circuits and so have more rounds and are much less efficient [20].
4. We construct the first distributed key generation protocol that requires only a *single round* to generate a key, after a one-time two-round initialization phase. To the best of our knowledge, the feasibility of achieving one-round key generation was unknown previously.
5. Our multiparty (probabilistic) Schnorr and distributed key generation protocols also have threshold variants, with the above number of rounds as long as the set of participating parties is known at the onset. These protocols fulfill a new property that we call **proof of quorum identity**: the participating parties can later prove that they are the ones who participated, but are unable to frame any other subset. We achieve this while still generating a full standard Schnorr signature.
6. We construct a hierarchical-deterministic (HD) key derivation method analogous to Bitcoin's BIP032 [38] that also enables parties to efficiently prove that a public key was derived correctly from the root secret. Unlike the standard BIP032, our derivation method is MPC friendly.

All of our protocols are UC secure [12] for static malicious adversaries and a dishonest majority, and are proven under standard assumptions in the random-oracle model.

The use of simulation-based MPC definition, like UC security [12], has many advantages. In particular, security under composition with any protocol is guaranteed, as well as security even

when related keys are used (like when BIP032 derivation is used) or when keys are generated with poor entropy. In such cases, the MPC protocol provides the same level of security as a locally computed signature, which is of course optimal. Our results provide an option to those who need two-round signing protocols, but still want to maintain full simulatability and composition.

We next provide a high-level description of our two eVRF schemes, both in the random-oracle model. Fix a "target group" $\mathbb{G}$ and a generator $G$ of $\mathbb{G}$. An eVRF is a triple $(\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Verify})$ of PPT algorithms, where (i) $\mathsf{KeyGen} \to (k, \mathsf{vk})$ samples the (secret) key $k$ and public verification key $\mathsf{vk}$, (ii) $\mathsf{Eval}(k, x) \to (y, Y, \pi)$ outputs a pseudorandom $y$, its value in the exponent $Y = y \cdot G$, and a proof $\pi$, and (iii) $\mathsf{Verify}(\mathsf{vk}, x, Y, \pi) \to \{0, 1\}$ verifies that $Y$ is consistent with $\mathsf{vk}$ and $x$.

**A construction from DDH.** A natural starting point is the classic DDH-based PRF [31] defined as $F(k, x) := H(x) \cdot G_{\mathrm{S}}$, where $G_{\mathrm{S}}$ is a generator of some (finite) cyclic group $\mathbb{G}_{\mathrm{S}}$, and $H$ is a hash function $H : \mathcal{X} \to \mathbb{G}_{\mathrm{S}}$. This PRF can proved secure when the DDH assumption holds in $\mathbb{G}_{\mathrm{S}}$ and $H$ is modeled as a random oracle. Concretely, $\mathbb{G}_{\mathrm{S}}$ can be the group of points of an elliptic curve $E$ defined over some prime field $\mathbb{F}_q$, where $q$ is the order of our target group $\mathbb{G}$. The eVRF will output $F(k, x)$ "in the exponent" of the target group $\mathbb{G}$. To do so we treat a point $P$ in the group $\mathbb{G}_{\mathrm{S}} = E(\mathbb{F}_q)$ as a pair $(x_P, y_P)$ in $\mathbb{F}_q^2$. Now, for a given key $k$ and input $x \in \mathcal{X}$ the evaluation algorithm $\mathsf{Eval}(k, x)$ works as follows:

1. Compute $P := k \cdot H(x)$ in $\mathbb{G}_{\mathrm{S}}$ and let $x_P \in \mathbb{F}_q$ be the $x$-coordinate of the point $P$;
2. Set $Y := x_P \cdot G$ in the target group $\mathbb{G}$;
3. Generate a zero-knowledge proof $\pi$ that $Y \in \mathbb{G}$ is computed correctly with respect to $x$ and a commitment to $k$.

Then $\mathsf{Eval}(k, x)$ outputs $(x_P, Y, \pi)$. The challenge is to design an efficient ZK proof that $Y$ is computed correctly. We design such a proof in Section 5. Our proof is concretely practical; we estimate that it takes only a few tens of milliseconds to generate and verify the proof on a single thread on a modern processor.

**A construction from Paillier.** Assume for a moment that the eVRF key owner has a secret trapdoor that lets it efficiently compute discrete logarithms in the target group $\mathbb{G}$. In such a case, we could let $H$ be a random oracle mapping arbitrary strings to uniform values in $\mathbb{G}$. Then, the eVRF evaluation would involve hashing the input $x$ into a random group element, $Y := H(x)$, and then computing $y = \log Y$. The verification procedure would simply verify that $H(x) = Y$. This would be a perfect eVRF, where every party would use a different group $\mathbb{G}$.

Since we have no trapdoors to enable the efficient computation of the discrete log in groups of interest, we take a similar approach using an intermediate hard problem for which the secret-key owner has a trapdoor. Specifically, let $H$ be a hash function $H : \mathcal{X} \mapsto [N]$, for some $N \geq |\mathbb{G}|$. Now, we could take any trapdoor permutation $f$ on the domain $[N]$, and have the secret key owner invert the permutation on $H(x)$ to get $y \in [N]$ and set $Y := y \cdot G$. It would then prove that $f^{-1}(H(x)) \cdot G = Y$. The challenge with this approach is finding an efficient zero-knowledge proof for this relation. To handle this challenge we use the homomorphic Paillier encryption scheme instead of a trapdoor permutation. The key generation algorithm outputs a Paillier public and secret key pair $(\mathsf{sk}, \mathsf{pk})$. The evaluation algorithm $\mathsf{Eval}(\mathsf{sk}, x)$ acts as follows:

1. Hash the input $x$ into a Paillier ciphertext $\widehat{Y}$, using a hash function $H : \mathcal{X} \to \mathbb{Z}_{\mathsf{pk}^2}$, where $\mathbb{Z}_{\mathsf{pk}^2}$ is the set of Pallier ciphertexts for $\mathsf{pk}$;
2. Decrypt $\widehat{Y}$ using $\mathsf{sk}$ to get the plaintext $y \in [\mathsf{pk}]$, and compute $Y := y \cdot G$ in $\mathbb{G}$;

3. Generate a zero-knowledge proof $\pi$ that the Pallier decryption of $\widehat{Y} \in \mathbb{Z}_{\mathsf{pk}^2}$ modulo $q$ is equal to the discrete log of $Y \in \mathbb{G}$ base $G$.

The value $y$ is pseudorandom since $\widehat{Y}$ is a random ciphertext ($H$ is modeled as a random oracle). Furthermore, since encryption is binding and the proof is sound, it is not possible to cheat and provide some different $Y' \neq Y$ and prove that it is consistent with $\mathsf{pk}$ and $x$. The only challenge is to efficiently compute the zero-knowledge proof. This proof can be made rather efficient, depending on the setting, as we show in Section 6.

**Overview of the rest of the paper.** After some preliminary definitions in Section 2, we formally define exponent VRFs in Section 3. We give two definitions: a game based definition and an ideal-functionality based definition. The game based definition is useful for constructing an eVRF, while the simulation based definition is useful for describing and proving security of the applications. Theorem 2 proves that the two definitions are equivalent (assuming a zero-knowledge proof-of-knowledge of the private key). In Section 4, we present the many applications of eVRFs and prove their security. Finally, our DDH-based eVRF is presented in Section 5, and the Paillier-based eVRF is presented in Section 6. Our work leaves a number of important open questions for future work described in Section 7.

## 2 Preliminaries

**Notation.** We use $\lambda \in \mathbb{Z}^{(>0)}$ to denote the security parameter, and $\overset{\text{c}}{\approx}$ to denote computational indistinguishability. We write $x \leftarrow y$ to denote the assignment of the value of $y$ to $x$, and $x \leftarrow\!\!\$\ S$ to denote sampling an element from the set $S$ independently and uniformly at random. Similarly, for a randomized algorithm $\mathcal{A}$, we write $y \leftarrow\!\!\$\ \mathcal{A}(x)$ to denote that $y$ is distributed according to the output of $\mathcal{A}(x)$ (over uniformly sampled random coins). We use $[n]$ for the set $\{1, \ldots, n\}$. We use additive notation for the group operation, and 0 for the group identity.

### 2.1 Pseudorandom Functions

We define pseudorandom functions and verifiable pseudorandom functions in a way that is convenient for the presentation in this paper. Since our eVRF constructions are given in the ROM, the following definitions are given for oracle-aided constructions. In such constructions, all entities (including the adversary) have oracle access to the same function, and they are secure in the ROM, if they are secure with respect to the all-function ensemble (the ensemble $\mathcal{O}$ according to the following definition). We start with formally defining ensemble of function families.

**Definition 1 (Function families).** *A **function family** with respect to domain/range $(\mathcal{X}, \mathcal{Y})$ is a family of functions $\mathcal{F} = \{f : \mathcal{X} \mapsto \mathcal{Y}\}$. We let $O_{\mathcal{X},\mathcal{Y}}$ denote the all-function family from $\mathcal{X}$ to $\mathcal{Y}$. A **function-family ensemble** with respect to domain/range ensemble $\{(\mathcal{X}_\lambda, \mathcal{Y}_\lambda)\}_{\lambda\in\mathbb{N}}$ is an ensemble of function families $\{\mathcal{F}_\lambda\}$, where each $\mathcal{F}_\lambda$ is a subset of $O_{\mathcal{X}_\lambda,\mathcal{Y}_\lambda}$.*

**Definition 2.** *A **pseudorandom function (PRF)** with respect to the domain/range ensemble $\{(\mathcal{X}_\lambda, \mathcal{Y}_\lambda)\}_{\lambda\in\mathbb{N}}$ and function-family ensemble $\mathcal{H}$, is a pair of oracle-aided* PPT *algorithms* (KeyGen, Eval) *such that for all $\lambda \in \mathbb{N}$ and $h \in \mathcal{H}_\lambda$:*

- $\mathsf{KeyGen}^h(1^\lambda) \to k$*: outputs a secret key $k \in \mathcal{K}$.*

– $\mathsf{Eval}^h(1^\lambda, k, x) \to y$: *on key* $k$ *and input* $x \in \mathcal{X}_\lambda$, deterministically *outputs* $y \in \mathcal{Y}_\lambda$.

*When clear from the context, we omit* $1^\lambda$ *from the input list of* Eval. *The PRF is* ***secure*** *if for all oracle-aided* PPT A*:*

$$\left|\Pr\left[\mathsf{A}^{h,\mathsf{Eval}_k^h(\cdot)}(1^\lambda) = 1\right] - \Pr\left[\mathsf{A}^{h,o(\cdot)}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda),$$

*where* $h \leftarrow\!\!\$\ \mathcal{H}_\lambda$, $k \leftarrow\!\!\$\ \mathsf{KeyGen}^h(1^\lambda)$, $o \leftarrow\!\!\$\ \mathcal{O}_{(\mathcal{X}_\lambda, \mathcal{Y}_\lambda)}$, *and* $\mathsf{Eval}_k^h(x) := \mathsf{Eval}^h(1^\lambda, k, x)$.

We next define verifiable pseudorandom functions (VRFs). Our definition strengthens the standard definition of VRFs, by (naturally) demanding *simulatability*, meaning that the verifiability proof does not leak significant information beyond correctness.

**Definition 3.** *A* ***simulatable verifiable random function (VRF)*** *with respect to domain/range ensemble* $\{(\mathcal{X}_\lambda, \mathcal{Y}_\lambda)\}_{\lambda\in\mathbb{N}}$ *and function-family ensemble* $\mathcal{H}$*, is a triple of oracle-aided* PPT *algorithms* $(\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Verify})$ *such that for every* $\lambda \in \mathbb{N}$ *and* $h \in \mathcal{H}_\lambda$*:*

– $\mathsf{KeyGen}^h(1^\lambda) \to (k, \mathsf{vk})$.

– $\mathsf{Eval}^h(1^\lambda, k, x) \to (y, \pi)$. *We let* $\mathsf{Eval}_1(1^\lambda, k, x) \to y$ *be the same as* Eval*, but only outputs its first output (i.e.,* $y$*).*

– $\mathsf{Verify}^h(1^\lambda, \mathsf{vk}, x, y, \pi) \to \{0, 1\}$.

*When clear from the context, we omit* $1^\lambda$ *from the inputs to* Eval *and* Verify. *The VRF is* ***secure*** *if*

– ***Correctness.*** *For all* PPT A*:*

$$\Pr\left[\neg\mathsf{Verify}^h(\mathsf{vk}, x, y, \pi)\right] \leq \mathsf{negl}(\lambda),$$

*where* $h \leftarrow\!\!\$\ \mathcal{H}_\lambda$, $(k, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{KeyGen}^h(1^\lambda)$, $x \leftarrow\!\!\$\ \mathsf{A}^{h,\mathsf{Eval}_k^h}(1^\lambda, \mathsf{vk})$, *and* $(y, \pi) \leftarrow\!\!\$\ \mathsf{Eval}_k^h(x)$.

– ***Pseudorandomness.*** $(\mathsf{KeyGen}, \mathsf{Eval}_1)$ *is a secure PRF with respect to the domain/range ensemble* $\{(\mathcal{X}_\lambda, \mathcal{Y}_\lambda)\}_{\lambda\in\mathbb{N}}$ *and* $\mathcal{H}$.

– ***Verifiability.*** *For all* PPT A*: if* $h \leftarrow\!\!\$\ \mathcal{H}_\lambda$ *and* $\big(\mathsf{vk}, x, (y_0, \pi_0), (y_1, \pi_1)\big) \leftarrow\!\!\$\ \mathsf{A}^h(1^\lambda)$ *then*

$$\Pr\big[\ y_0 \neq y_1 \quad \wedge \quad (\forall i \in \{0,1\} \colon \mathsf{Verify}^h(\mathsf{vk}, x, y_i, \pi_i) = 1)\big] \leq \mathsf{negl}(\lambda).$$

– ***Simulatability.*** *There exists a* PPT *Sim such that for all* $x \in \mathcal{X}_\lambda$*:*

$$\big(\mathsf{vk}, k, x, \mathit{Sim}^h(\mathsf{vk}, x, y)\big) \quad \stackrel{c}{\approx} \quad \big(\mathsf{vk}, k, x, \mathsf{Eval}^h(k, x)\big)$$

*for* $h \leftarrow\!\!\$\ \mathcal{H}_\lambda$, $(k, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{KeyGen}^h(1^\lambda)$, *and* $y \leftarrow \mathsf{Eval}_1^h(k, x)$.

### 2.2 Secure Computation

For our ideal-model definition of exponent VRFs and for our applications, we prove security for the stand-alone definition of secure multiparty computation [11,22] for security with abort (where some honest parties may have output and some may abort) and with no honest majority. In this model, all parties send their inputs to the ideal functionality (computed by a trusted party). The ideal functionality then sends the (ideal-model) adversary the corrupted parties' outputs, and the adversary then instructs the ideal functionality as to which honest parties should receive output.

We denote the set of honest parties sent by the ideal adversary to the ideal functionality to receive output by $\mathcal{O}_{\mathsf{out}}$.

Although we prove security in the stand-alone model that guarantees security under sequential composition only, we are really interested in UC security [12]; i.e., security under concurrent general composition. This is achieved by all our protocols since they all have *straight-line simulation* (i.e., no rewinding). As shown in [25], this implies UC security if the protocol is *perfectly secure* or there is *start synchronization* (meaning that all parties have their input before the protocol begins).

**Network model.** In all of our protocols, we consider security with abort. As such, parties can just wait to receive a message, and "hang" if they do not (in practice, they can just abort if they wait too long, which is also fine). This means that we don't need to assume a synchronous network, as parties proceed to the next round only after receiving all messages from the previous round.

## 3 eVRFs

In this section, we formally define the concept of an eVRF. We begin by defining a game-based definition for the security of an eVRF. Next we define an eVRF ideal functionality, and prove that a simple protocol using the game-based definition, together with a zero-knowledge proof of knowledge of the private key, securely realizes the ideal functionality. The game-based definition will be used to argue that our eVRF constructions are secure, and the ideal functionality will be used for constructing our applications utilizing an eVRF.

### 3.1 Game-based Definition

Let $\mathbb{G}$ be a finite cyclic group with generator $G \in \mathbb{G}$. The evaluation algorithm $\mathsf{Eval}(k, x)$ of an eVRF outputs a triple $(y, Y, \pi)$ such that $Y = y \cdot G$, with the property that $\mathsf{Eval}_1(k, x) := y$ is a pseudorandom function, and $\mathsf{Eval}_2(k, x) := (Y, \pi)$ is a (simulatable) VRF. Stated differently, the output $y$ is pseudorandom, and there exists a (simulatable) proof that $Y$ has been generated correctly from $y$ as $Y \leftarrow y \cdot G$. The formal game-based definition, suited for constructions in the ROM, is given below.

**Definition 4.** *Let $\{(\mathcal{X}_\lambda, \mathcal{Y}_\lambda)\}_{\lambda\in\mathbb{N}}$ be an ensemble of domains/ranges, where each $\mathcal{Y}_\lambda$ is a finite cyclic group with a specified generator $G_\lambda$. An **exponent verifiable random function (eVRF)** with respect to domain/range ensemble $\{(\mathcal{X}_\lambda, \mathcal{Y}_\lambda)\}_{\lambda\in\mathbb{N}}$ and function-family ensemble $\mathcal{H}$, is a triple of oracle-aided* PPT *algorithms called* $(\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Verify})$ *such that for every $\lambda \in \mathbb{N}$ and $h \in \mathcal{H}_\lambda$:*

- $\mathsf{KeyGen}^h(1^\lambda) \to (k, \mathsf{vk})$.
- $\mathsf{Eval}^h(1^\lambda, k, x) \to (y, Y, \pi)$ *with $x \in \mathcal{X}_\lambda$, $y \in \mathbb{Z}_{|\mathcal{Y}_\lambda|}$, and $Y \in \mathcal{Y}_\lambda$. We define two auxiliary algorithms $\mathsf{Eval}_1(1^\lambda, k, x) \to y$ and $\mathsf{Eval}_2(1^\lambda, k, x) \to (Y, \pi)$ that are the same as* Eval*, but only output the first output (i.e., $y$) or the second and third outputs (i.e., $(Y, \pi)$) of* Eval*, respectively.*
- $\mathsf{Verify}^h(1^\lambda, \mathsf{vk}, x, Y, \pi) \to \{0, 1\}$.

*When clear from the context, we omit $1^\lambda$ from the inputs to* Eval *and* Verify*. An eVRF is **secure** if*

- ***Consistency.** For every* PPT A*:*

$$\Pr\left[ y \cdot G_\lambda \neq Y \ : \ \begin{array}{l} h \leftarrow\!\!\$\ \mathcal{H}_\lambda, \quad (k, \mathsf{vk}) \leftarrow \mathsf{KeyGen}^h(1^\lambda) \\ x \leftarrow\!\!\$\ \mathsf{A}^{h, \mathsf{Eval}^h(k,\cdot)}(1^\lambda, \mathsf{vk}), \quad (y, Y, \pi) \leftarrow\!\!\$\ \mathsf{Eval}^h(k, x) \end{array} \right] \leq \mathsf{negl}(\lambda).$$

– ***Pseudorandomness.*** $(\mathsf{KeyGen}, \mathsf{Eval}_1)$ *is a secure PRF with respect to the domain/range ensemble* $\left\{\left(\mathcal{X}_\lambda, \mathbb{Z}_{|\mathcal{Y}_\lambda|}\right)\right\}_{\lambda\in\mathbb{N}}$ *and* $\mathcal{H}$.
– ***Simulatable verifiability.*** $(\mathsf{KeyGen}, \mathsf{Eval}_2, \mathsf{Verify})$ *is a simulatable VRF with respect to the ensemble* $\{(\mathcal{X}_\lambda, \mathcal{Y}_\lambda)\}_{\lambda\in\mathbb{N}}$ *and* $\mathcal{H}$.

### 3.2 Ideal Definition

We now define the ideal functionality for an eVRF and prove that it is implied by the game-based definition, together with a zero-knowledge proof of knowledge of the private key. To simplify the notation we refer to an explicit domain/range $(\mathcal{X}, \mathcal{Y})$ rather than an ensemble.

**Definition 5 (eVRF functionality).** *Let* $(\mathcal{X}, \mathcal{Y})$ *be as in Definition 4, where* $\mathcal{Y}$ *defines a group* $\mathbb{G}$ *of order* $q$ *with generator* $G$. *The* ***eVRF ideal functionality*** *for* $(\mathcal{X}, \mathcal{Y})$, *denoted* $\mathcal{F}_{\mathsf{eVRF}}^{\mathcal{X},\mathcal{Y}}$ *or just* $\mathcal{F}_{\mathsf{eVRF}}$ *for short, is defined as follows:*

1. *Upon receiving* $(\mathsf{init}, i, *)$ *from some* $P_i$.
   (a) *If* $P_i$ *is honest and the input is* $(\mathsf{init}, i)$, *then receive a value* $\mathsf{sid}$ *from the (ideal) adversary, verify that it's unique and store* $(\mathsf{sid}, i)$
   (b) *If* $P_i$ *is corrupted and the input is* $(\mathsf{init}, i, \mathsf{sid}, f)$ *where* $f$ *is the description of a deterministic polynomial-time computable function* $\mathsf{sid}$ *has not been stored, then store* $(\mathsf{sid}, i, f)$
   *Send* $(\mathsf{init}, i, \mathsf{sid})$ *to all parties*
2. *Upon receiving* $(\mathsf{eval}, i, \mathsf{sid}, x)$ *from* $P_i$, *where* $x \in \mathcal{X}$:
   (a) *If* $(\mathsf{sid}, i)$ *or* $(\mathsf{sid}, i, f)$ *is not stored then ignore*
   (b) *If* $P_i$ *is honest:*
      i. *If there does not exist a stored tuple* $(\mathsf{sid}, i, x, y, Y)$ *with* $x$, *then choose a random* $y \leftarrow\!\!\$\ \mathbb{Z}_q$, *compute* $Y \leftarrow y \cdot G$ *and store* $(\mathsf{sid}, i, x, y, Y)$
      ii. *Send* $(\mathsf{eval}, i, \mathsf{sid}, x, y, Y)$ *to party* $P_i$ *and* $(\mathsf{eval}, i, \mathsf{sid}, x, Y)$ *to all parties*
   (c) *If* $P_i$ *is corrupted, then compute* $Y \leftarrow f(x) \cdot G$ *and send* $(\mathsf{eval}, i, \mathsf{sid}, x, Y)$ *to all parties*

In order to prove that the game-based definition implies the ideal functionality, we define a protocol that utilizes the game-based definition, and then prove that it securely realizes $\mathcal{F}_{\mathsf{eVRF}}$. Our protocol requires a ZK proof of knowledge for relation $\mathcal{R}_{EF} = \left\{(\mathsf{vk}, (k, r)) \mid \exists r : \mathsf{KeyGen}(1^\lambda; r) = (k, \mathsf{vk})\right\}$. We denote the ideal functionality for this proof by $\mathcal{F}_{\mathsf{zk}}^{\mathcal{R}_{EF}}$; the functionality receives $(\mathsf{prove}, i, j, \mathsf{vk}, k, r)$ from $P_i$ and sends $(\mathsf{prove}, i, j, \mathsf{vk}, 1)$ to $P_j$ if $(\mathsf{vk}, (k, r)) \in \mathcal{R}_{EF}$. We note that by including this proof of knowledge, it is not possible for a party to copy an eVRF instance from another party (practically, the identity $i$ of the prover is just included in the hash in the proof).

The ideal functionality $\mathcal{F}_{\mathsf{eVRF}}$ needs an $\mathsf{sid}$ in order to distinguish different eVRF instances. In the protocol, we use the verification key $\mathsf{vk}$ for this purpose and so do not require any additional $\mathsf{sid}$.

**The $\pi_{EF}$ protocol.** For any eVRF $EF = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Verify})$, we define the protocol $\pi_{EF}$ with parameters $(\mathcal{X}, \mathcal{Y})$ as follows:

**Protocol 1 ($\pi_{EF}$)**

– ***Initialize:***
   1. *Message 1 from* $P_i$: *Party* $P_i$ *with input* $(\mathsf{init}, i)$,
      (a) $(k, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{X}, \mathcal{Y})$
      (b) *Send* $(\mathsf{init}, i, \mathsf{vk})$ *to all parties* $P_1, \ldots, P_n$

(c) *Send* $(\mathsf{prove}, i, j, \mathsf{vk}, k, r)$ *to* $\mathcal{F}_{\mathsf{zk}}^{\mathcal{R}_{EF}}$ *for all* $j \in [n]$

2. *Message 2: Each party* $P_j$ *upon receiving* $(\mathsf{init}, i, \mathsf{vk})$ *from party* $P_i$
   (a) *If* $(\mathsf{prove}, i, j, \mathsf{vk}, 1)$ *is received from* $\mathcal{F}_{\mathsf{zk}}^{\mathcal{R}_{EF}}$ *then proceed; else ignore*
   (b) *Send* $(\mathsf{init}, i, \mathsf{vk})$ *to all parties* $P_1, \ldots, P_n$
3. *Output: Upon receiving* $(\mathsf{init}, i, \mathsf{vk})$ *from all parties,*
   (a) *Party* $P_i$*: Output* $(\mathsf{init}, i, \mathsf{vk}, k)$
   (b) *All other parties* $P_j$ $(j \neq i)$*: if the same message* $(\mathsf{init}, i, \mathsf{vk})$ *is received from all parties then store* $(\mathsf{init}, i, \mathsf{vk})$*; else ignore*

– ***Evaluate:***

1. *Message from* $P_i$*: Party* $P_i$ *with input* $(\mathsf{eval}, i, \mathsf{vk}, x)$*,*
   (a) $(y, Y, \pi) \leftarrow \mathsf{Eval}(k, x)$
   (b) *Send* $(\mathsf{eval}, i, \mathsf{vk}, x, Y, \pi)$ *to all parties* $P_1, \ldots, P_n$
2. *Output:*
   (a) *Party* $P_i$*: Output* $(\mathsf{eval}, i, \mathsf{vk}, x, y, Y)$
   (b) *All other parties* $P_j$ $(j \neq i)$*: Upon receiving* $(\mathsf{eval}, i, \mathsf{vk}, x, Y, \pi)$ *from* $P_i$
      i. *Verify that* $(\mathsf{init}, i, \mathsf{vk})$ *has been stored (i.e.,* $\mathsf{vk}$ *is associated with* $P_i$*)*
      ii. *If* $\mathsf{Verify}(\mathsf{vk}, x, Y, \pi) = 0$ *then ignore*
      iii. *Else, output* $(\mathsf{eval}, i, \mathsf{vk}, x, Y)$

We stress that in the *two-party case*, the initialize phase consists only of $P_i$ sending $(\mathsf{init}, i, \mathsf{vk})$ to $P_j$, who stores it (i.e., there is no need for a second message in order to obtain consensus). We now prove that $\pi_{EF}$ securely realizes $\mathcal{F}_{\mathsf{eVRF}}$.

**Theorem 2.** *Let* $EF = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Verify})$ *be an exponent verifiable random function with respect to* $(\mathcal{X}, \mathcal{Y})$*. Then* $\pi_{EF}$ *securely realizes with abort* $\mathcal{F}_{\mathsf{eVRF}}$ *with respect to* $(\mathcal{X}, \mathcal{Y})$ *in the* $\mathcal{F}_{\mathsf{zk}}^{\mathcal{R}_{EF}}$*-hybrid model, in the presence of a static malicious adversary corrupting any number of parties.*

*Proof.* Let $P_1, \ldots, P_n$ be the parties, let $\mathcal{I} \subset [n]$ be the set of corrupted parties, and let $\mathcal{A}$ be a real-world adversary running protocol $\pi_{EF}$. We construct an ideal-model adversary/simulator $\mathcal{S}$ as follows:

1. **Initialize:**
   (a) Upon receiving $(\mathsf{init}, i)$ from $\mathcal{F}_{\mathsf{eVRF}}$ for an honest $P_i$, the simulator $\mathcal{S}$ runs $\mathsf{KeyGen}(1^\lambda, \mathcal{X}, \mathcal{Y})$ to obtain $(k, \mathsf{vk})$, and sends $\mathsf{sid} = \mathsf{vk}$ to $\mathcal{F}_{\mathsf{eVRF}}$. Next, $\mathcal{S}$ simulates honest party $P_i$ sending $(\mathsf{init}, i, \mathsf{vk})$ to all corrupted parties, and simulates $\mathcal{F}_{\mathsf{zk}}^{\mathcal{R}_{EF}}$ sending $(\mathsf{prove}, i, j, \mathsf{vk}, 1)$ to all corrupted parties. $\mathcal{S}$ then simulates message 2 of the initialization protocol and defines $\mathcal{O}_{\mathsf{out}}$ to be the set of honest parties who would not abort (i.e., who all received the same correct messages from all corrupted parties). $\mathcal{S}$ sends $\mathcal{O}_{\mathsf{out}}$ to $\mathcal{F}_{\mathsf{eVRF}}$, instructing it to send output to the honest parties in $\mathcal{O}_{\mathsf{out}}$.
   (b) Upon a corrupted party receiving $(\mathsf{init}, i)$ for input, $\mathcal{S}$ obtains the message $(\mathsf{init}, i, \mathsf{vk})$ sent by $\mathcal{A}$ to all honest parties and the messages $(\mathsf{prove}, i, j, \mathsf{vk}, k, r)$ sent to $\mathcal{F}_{\mathsf{zk}}^{\mathcal{R}_{EF}}$ for all $j \notin \mathcal{I}$. Then, $\mathcal{S}$ simulates the honest parties actions exactly according to the protocol, and defines $\mathcal{O}_{\mathsf{out}}$ to be the set of honest parties who would not abort (i.e., who all received the same correct messages from all corrupted parties, and who received correct proofs). If $\mathcal{O}_{\mathsf{out}}$ is not empty, then $\mathcal{S}$ defines $f(x) = \mathsf{Eval}_1(k, x)$ where $k$ is from the message sent by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{zk}}^{\mathcal{R}_{EF}}$, and sends $(\mathsf{init}, \mathsf{vk}, i, f)$ to $\mathcal{F}_{\mathsf{eVRF}}$ together with $\mathcal{O}_{\mathsf{out}}$, instructing it to send output to the honest parties in $\mathcal{O}_{\mathsf{out}}$.

2. **Evaluate:**
   (a) Upon receiving $(\mathsf{eval}, i, \mathsf{vk}, x, Y)$ from $\mathcal{F}_{\mathsf{eVRF}}$ for an honest $P_i$ ($\mathsf{vk}$ is the $\mathsf{sid}$ as generated during initialize), the simulator $\mathcal{S}$ runs $\mathsf{Sim}(\mathsf{vk}, x, Y)$ to obtain $\pi$ and simulates the honest $P_i$ sending $(\mathsf{eval}, i, \mathsf{vk}, x, Y, \pi)$ to all corrupted parties for the associated $\mathsf{vk}$.
   (b) Upon a corrupted party receiving $(\mathsf{eval}, i, \mathsf{vk}, x)$ for input (as above, $\mathsf{vk}$ is the $\mathsf{sid}$), $\mathcal{S}$ sends $(\mathsf{eval}, i, \mathsf{vk}, x)$ to $\mathcal{F}_{\mathsf{eVRF}}$ and obtains the messages $\left\{(\mathsf{eval}, i, \mathsf{vk}, x, Y^j, \pi^j)\right\}_{j \notin \mathcal{I}}$ sent by $\mathcal{A}$ to the honest parties $P_j$ for $j \notin \mathcal{I}$; observe that nothing forces $\mathcal{A}$ to send the same message to all parties and so we denote by $(Y^j, \pi^j)$ denote the values received by honest $P_j$.[5] For each $j \notin \mathcal{I}$, $\mathcal{S}$ verifies that $\mathsf{Verify}(\mathsf{vk}, x, Y^j, \pi^j) = 1$. If no, it ignores the message. Else, it adds $P_j$ to $\mathcal{O}_{\mathsf{out}}$, and sends $\mathcal{O}_{\mathsf{out}}$ to $\mathcal{F}_{\mathsf{eVRF}}$.

We separately consider the case that $P_i$ is honest and that $P_i$ is corrupted.

Let $P_i$ be honest. Then, the simulation in the initalize phase is perfect (in the $\mathcal{F}_{\mathsf{zk}}^{\mathcal{R}_{EF}}$-hybrid model) since $\mathcal{S}$ generates $(k, \mathsf{vk})$ like an honest party and perfectly simulates the message from $\mathcal{F}_{\mathsf{zk}}^{\mathcal{R}_{EF}}$ to the corrupted parties. Regarding the evaluation phase, there are two differences between the simulation and a real execution: **(a)** the value $Y$ is truly random in the ideal execution (and in particular is independent of $(k, \mathsf{vk})$ generated by $\mathcal{S}$ in the initialization phase) and equals $\mathsf{Eval}_1(k, x) \cdot G$ in the real execution, and **(b)** the proof $\pi$ is simulated in the ideal execution and is output from $\mathsf{Eval}(k, x)$ in the real execution. We prove indistinguishability in three hybrid steps:

1. *Hybrid 1:* In this hybrid execution, we modify $\mathcal{F}_{\mathsf{eVRF}}$ so that upon receiving $(\mathsf{init}, i)$ from an honest $P_i$ it computes $\mathsf{KeyGen}(1^\lambda, \mathcal{X}, \mathcal{Y})$ to obtain $(k, \mathsf{vk})$, and sends $(\mathsf{init}, i, \mathsf{vk})$ to $\mathcal{S}$ setting $\mathsf{sid} = \mathsf{vk}$, instead of receiving $\mathsf{sid}$ from $\mathcal{S}$. Furthermore, $\mathcal{S}$ uses $\mathsf{vk}$ as it received from $\mathcal{F}_{\mathsf{eVRF}}$ instead of generating it by itself. Everything else remains the same, and in particular the evaluation is random.
   It is clear that the output distributions of this hybrid and the ideal execution are identical. The only difference is who chooses $\mathsf{vk}$, which makes no difference.
2. *Hybrid 2:* In this hybrid execution, we further modify $\mathcal{F}_{\mathsf{eVRF}}$ so that upon receiving $(\mathsf{eval}, i, \mathsf{sid}, x)$ from an honest $P_i$, instead of choosing a random $y$, it computes $y \leftarrow \mathsf{Eval}_1(k, x)$ with the $k$ generated after receiving $(\mathsf{init}, i)$. (The computation of $Y \leftarrow y \cdot G$ is unchanged.) Everything else remains the same as the first hybrid, including $\mathcal{S}$.
   The only difference between the first and second hybrid execution is how $y$ is generated. In order to show that this is indistinguishable, we rely on the fact that $\mathsf{Eval}_1$ is a pseudorandom function. Observe that it is possible to simulate both hybrid executions without ever receiving $k$ (in particular, because the proof $\pi$ is simulated). Thus, if it is possible to distinguish between hybrid 2 and hybrid 1, then it is possible to distinguish $\mathsf{Eval}_1$ from random.
3. *Hybrid 3:* In this hybrid execution, we modify $\mathcal{F}_{\mathsf{eVRF}}$ so that upon receiving $(\mathsf{eval}, i, \mathsf{sid}, x)$, instead of just sending $(\mathsf{eval}, i, \mathsf{sid}, x, Y)$ to $\mathcal{S}$, it also sends $\pi$ where $\mathsf{Eval}(k, x) = (y, Y, \pi)$. The simulator $\mathcal{S}$ is the same as for hybrid 2 except that instead of computing $\mathsf{Sim}(\mathsf{vk}, x, Y)$ to obtain $\pi$, it sends the proof $\pi$ received from $\mathcal{F}_{\mathsf{eVRF}}$. Indistinguishability of hybrid 2 and hybrid 3 follows from Definition 4 and in particular that $(\mathsf{vk}, k, x, \mathsf{Sim}(\mathsf{vk}, x, Y)) \overset{c}{\approx} (\mathsf{vk}, k, x, \mathsf{Eval}_2(k, x)))$. We stress that in order to simulate these hybrids, it is necessary to know $k$ (and the distinguisher only receives $\pi$ externally and needs to determine which distribution it is from). However, this is fine since the indistinguishability of a simulated and real proof holds even if $k$ is known.

[5] We consider different $Y^j, \pi^j$ values but not different $x$ values. This is because a different $x$ is considered a different evaluation and is treated separately.

Finally, we observe that hybrid 3 is identical to a real execution. The only difference is that $\mathcal{F}_{\mathsf{eVRF}}$ computes the honest parties' messages according to protocol $\pi_{EF}$ instead of the parties themselves, but this does not affect the output distribution.

We now consider the case that $P_i$ is corrupted. In this case, the simulator $\mathcal{S}$ perfectly detects who will receive output and who not in terms of receiving consistent messages during initialization and a valid proof via $\mathcal{F}_{\mathsf{zk}}^{\mathcal{R}_{EF}}$. Furthermore, $\mathcal{S}$ perfectly detects who will receive output in the evaluation phase, based on the proof $\pi$ being valid. However, in the ideal execution, the output received by honest parties during evaluation is always $f(x) = \mathsf{Eval}_1(k, x)$. Thus, there can only be a difference between the ideal and real executions if $\mathcal{A}$ sends a value $Y' \neq \mathsf{Eval}_1(k, x) \cdot G$ together with an *accepting proof* $\pi$. This would contradict the verifiability property of $(\mathsf{KeyGen}, \mathsf{Eval}_2, \mathsf{Verify})$ as a VRF. In particular, it is always possible to generate an accepting proof for $Y = \mathsf{Eval}_1(k, x) \cdot G$. If $\mathcal{A}$ generates an accepting proof also for some $Y' \neq \mathsf{Eval}_1(k, x) \cdot G$, then we could construct an adversary who outputs two distinct $Y, Y'$ with respective accepting proofs $\pi, \pi'$. This completes the proof. □

**UC security.** The simulator in the proof of Theorem 2 is straight line. Therefore, by [25], the protocol is UC secure assuming start synchronization (all parties have their input before the protocol starts). However, in both the initialize and evaluate phases, the only party with input is $P_i$ and therefore start synchronization holds always. We therefore have:

**Corollary 1.** *Let $EF = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Verify})$ be an exponent verifiable random function by Definition 4. Then $\pi_{EF}$ UC realizes $\mathcal{F}_{\mathsf{eVRF}}$ with abort in the $\mathcal{F}_{\mathsf{zk}}^{\mathcal{R}_{EF}}$-hybrid model, in the presence of a static malicious adversary corrupting any number of parties.*

## 4 Applications

In this section, we present the many applications for eVRFs discussed in the introduction. All the protocols we present are "fully simulatable" meaning that they securely realize the *plain algorithm functionality* (e.g., Schnorr signing), as opposed to some modified functionality. All of our protocols are concretely efficient, and are secure under standard assumptions in the random-oracle model.

### 4.1 One-Round Simulatable Distributed Key Generation

This protocol works by defining each party's key share to be the result of a pseudorandom function applied to a unique nonce. In order to ensure that each party uses its committed pseudorandom function, we use the $\mathcal{F}_{\mathsf{eVRF}}$ functionality to derive each party's key share. Intuitively, this enables us to generate a key in a single round since the only message the parties need to send is their single eVRF value. This suffices since each party can simply sum the public key-share values (we call $K_i = k_i \cdot G$ a party's public share) to obtain the final public key. The crucial difference between this key generation and standard key generation protocols is that since each party is already committed to its value via the $\mathcal{F}_{\mathsf{eVRF}}$ (after running a single initialization step), the corrupted parties cannot bias the output key. In particular, if a key was generated by each party simply choosing a random $k_i$ and sending $K_i = k_i \cdot G$ to all other parties, then a single corrupted party $P_1$ can completely determine the key by obtaining all the honest parties shares $K_2, \ldots, K_n$ and then setting $K_1 = K - \sum_{i=2}^{n} K_i$, where $P_1$ has chosen $K = k \cdot G$ where $k$ is known to it. This trivial attack can be prevented by having each party add a zero-knowledge proof of knowledge of the discrete log of its

$K_i$; this would prevent $P_1$ from carrying out this attack since it cannot know the discrete log of $K_1$. However, the protocol is still not simulatable since $P_1$ could bias the output. In particular, if $P_1$ wanted the public key $K$ to have a certain property that holds in 1/1000 keys then it can receive $K_2, \ldots, K_n$ and then repeatedly choose random $k_1$ and compute $K_1 = k_1 \cdot G$ and $K = \sum_{i=1}^{n} K_i$ until $K$ has the required property. All of this isn't possible with our protocol since the corrupted parties are committed to the PRF output via $\mathcal{F}_{\mathsf{eVRF}}$.

We remark that generating a new key requires a unique $\mathsf{nonce}$. In particular, if the same $\mathsf{nonce}$ is used twice then the same key will be output. This holds in both the ideal and real models, and therefore the security of the protocol does not rely on the nonce necessarily being unique. However, using the protocol to generate a new key does require ensuring a unique nonce (which could be a timestamp, a counter, etc.).

**The additive DKG functionality $\mathcal{F}_{\mathsf{dkg}}$:** Let $\mathbb{G}$ be a group of order $q$ with generator $G$. The distributed key-generation functionality $\mathcal{F}_{\mathsf{dkg}}$ for $\mathbb{G}$ running with parties $P_1, \ldots, P_n$ and corrupted parties $\mathcal{I} \subseteq [n]$ is defined as follows:

1. Wait to receive $(\mathsf{gen}, \mathsf{nonce})$ from all honest parties and $(\mathsf{gen}, \mathsf{nonce}, k_i)$ from all corrupted parties $P_i$ with $i \in \mathcal{I}$
2. If $(\mathsf{gen}, \mathsf{nonce}, k_1, \ldots, k_n)$ has already been stored
   – Retrieve $k_1, \ldots, k_n$
   Else
   – Choose random $k_j \leftarrow\!\!\$\, \mathbb{Z}_q$ for every $j \in [n] \setminus \mathcal{I}$
   – Store $(\mathsf{gen}, \mathsf{nonce}, k_1, \ldots, k_n)$
3. For $i = 1, \ldots, n$, compute $K_i = k_i \cdot G$
4. Compute $K = \sum_{i=1}^{n} K_i$
5. For $i = 1, \ldots, n$, send $(\mathsf{gen}, \mathsf{nonce}, k_i, K_1, \ldots, K_n, K)$ to $P_i$

We are now ready to present the protocol. Let $\mathcal{X} = \{0,1\}^\lambda$ and let $\mathcal{Y} = \mathbb{G}$ as desired for $\mathcal{F}_{\mathsf{dkg}}$.

**Protocol 3 ($\Pi_{\mathsf{dkg}}$ for additive DKG)**

– ***Initialize:***
  1. *In parallel, each $P_i$ sends $(\mathsf{init}, i)$ to $\mathcal{F}_{\mathsf{eVRF}}$ (for $\mathcal{X}, \mathcal{Y}$)*
  2. *Wait to receive $(\mathsf{init}, j, \mathsf{sid}_j)$ from $\mathcal{F}_{\mathsf{eVRF}}$ for all $j \in [n]$*
– ***Generate (one round):*** *upon input $(\mathsf{gen}, \mathsf{nonce})$, each party $P_i$*
  • Message:
    1. *Send $(\mathsf{eval}, i, \mathsf{sid}_i, \mathsf{nonce})$ to $\mathcal{F}_{\mathsf{eVRF}}$ and receive back $(\mathsf{eval}, i, \mathsf{sid}_i, \mathsf{nonce}, k_i, K_i)$*
  • Output:
    1. *Wait to receive $(\mathsf{eval}, j, \mathsf{sid}_j, \mathsf{nonce}, K_j)$ from $\mathcal{F}_{\mathsf{eVRF}}$ for all parties $P_j$ (all with the correct $\mathsf{nonce}$)*
    2. *Compute $K = \sum_{i=1}^{n} K_i$*
    3. *Output $(\mathsf{gen}, \mathsf{nonce}, k_i, K_1, \ldots, K_n, K)$*

The rationale behind the security of the protocol has already been described above and so we proceed directly to the proof of security.

**Theorem 4.** *Protocol 3 securely realizes $\mathcal{F}_{\mathsf{dkg}}$ with perfect security with abort, in the presence of a static malicious adversary corrupting up to n parties. Furthermore, after a single two-round initialization phase, each generation consists of a single round.*

*Proof.* If all $n$ parties or no parties are corrupted, then the statement is trivial (if no parties are corrupted, then since the ideal functionality and honest parties communicate over ideal private channels, nothing is revealed). Let $\mathcal{A}$ be the adversary and let $\mathcal{I}$ be the set of corrupted parties with $0 < |\mathcal{I}| < n$. We construct a simulator $\mathcal{S}$ with "internal communication" to $\mathcal{A}$ and "external communication" to $\mathcal{F}_{\mathsf{dkg}}$, as follows:

- *Initialize:*
  1. Simulate $\mathcal{F}_{\mathsf{eVRF}}$ sending $\{(\mathsf{init}, j)\}_{j \notin \mathcal{I}}$ to $\mathcal{A}$, and receive back $\{\mathsf{sid}_j\}_{j \notin \mathcal{I}}$ as $\mathcal{A}$ would send to $\mathcal{F}_{\mathsf{eVRF}}$
  2. Simulate $\mathcal{F}_{\mathsf{eVRF}}$ sending $(\mathsf{init}, j, \mathsf{sid}_j)$ to all parties, for all $j \notin \mathcal{I}$
  3. For all $i \in \mathcal{I}$, internally receive $(\mathsf{init}, i, \mathsf{sid}_i, f_i)$ from $\mathcal{A}$ as sent to $\mathcal{F}_{\mathsf{eVRF}}$ from $P_i$, and simulate $\mathcal{F}_{\mathsf{eVRF}}$ sending back $(\mathsf{init}, i, \mathsf{sid}_i)$ to all parties
- *Generate:* to simulate an execution of generate for $\mathsf{nonce}$,
  1. For every $i \in \mathcal{I}$, compute $k_i = f_i(\mathsf{nonce})$, where $f_i$ is as received for $P_i$ in the initialize phase
  2. Send $(\mathsf{gen}, \mathsf{nonce}, k_i)$ externally to $\mathcal{F}_{\mathsf{dkg}}$ for every $i \in \mathcal{I}$, and receive back $\{(\mathsf{gen}, \mathsf{nonce}, k_i, K_1, \ldots, K_n, K)\}_{i \in \mathcal{I}}$
  3. Simulate $\mathcal{F}_{\mathsf{eVRF}}$ sending $(\mathsf{eval}, j, \mathsf{sid}_j, \mathsf{nonce}, K_j)$ to all corrupted parties, for every $j \in [n] \setminus \mathcal{I}$
  4. Define the set $\mathcal{O}_{\mathsf{out}}$ of honest parties to receive output by the set of parties for which $\mathcal{A}$ instructs $\mathcal{F}_{\mathsf{eVRF}}$ to provide output from all $i \in \mathcal{I}$
  5. Send $\mathcal{F}_{\mathsf{dkg}}$ the set $\mathcal{O}_{\mathsf{out}}$
  6. Output whatever $\mathcal{A}$ outputs

The initialization phase in the simulation is identical to the real execution in the $\mathcal{F}_{\mathsf{eVRF}}$-hybrid model. Regarding the generate phase, in the $\mathcal{F}_{\mathsf{eVRF}}$-hybrid model, the view of the adversary in the ideal execution is also identical to its view in a real execution. This is because in the $\mathcal{F}_{\mathsf{eVRF}}$ model all that it sees is the $K_j$ values for the honest parties. Furthermore, these values are uniformly distributed in the real execution by the definition of $\mathcal{F}_{\mathsf{eVRF}}$, and uniformly distributed in the ideal execution by the definition of $\mathcal{F}_{\mathsf{dkg}}$. Finally, the $k_i$ values of the corrupted parties is the same, since $\mathcal{S}$ sends $\mathcal{F}_{\mathsf{dkg}}$ the same values that $\mathcal{A}$ is committed to by the definition of $\mathcal{F}_{\mathsf{eVRF}}$.

The round complexity statement in the theorem follows by observation that each $\mathcal{F}_{\mathsf{eVRF}}$ initialization operation is two rounds and each evaluation operation is just a single round (as described in Protocol 1, and these can all be sent in parallel. This completes the proof. □

### 4.2 One-Round Simulatable Threshold Distributed Key Generation

The protocol in Section 4.1 works for a set of $n$ parties who all participate and wish to generate a key that is additively distributed amongst themselves. This can easily be extended to a threshold setting (and even to a more general access structure of a tree of AND, OR, and threshold gates) by simply having each party in a quorum generate a VSS sharing of its key share defined by the eVRF. This would work but would not be a single round only since a consensus round would be needed to ensure that all parties receive the same sharing (a simple echo-broadcast suffices, as shown in [14]). Fortunately, we can use the eVRF to achieve this as well, and have each party define *all the coefficients* in its polynomial for Feldman VSS [17] via the eVRF. Since each party is already committed to its values and therefore its polynomial via the eVRF, and since all parties can verify that the eVRF output values sent are correct, there is no need for an additional consensus round. Indeed, no party can send a value that isn't correct. We describe the protocol for a simple threshold only; the extension to a tree of AND, OR and threshold gates is immediate.

We describe the protocol with a quorum of $t+1$ online parties who generate the key for all $n$ parties. We stress that there is no security benefit in having all $n$ parties generate the key, since any quorum of $t+1$ parties will anyway have the entire key.

**The quorum-specific threshold DKG functionality $\mathcal{F}_{\mathsf{dkg}\mathcal{Q}}^{n,t}$:** Let $\mathbb{G}$ be a group of order $q$ with generator $G$, let $\alpha_1, \ldots, \alpha_n$ be fixed distinct elements in $\mathbb{Z}_q$, and let $t < n$. The distributed key-generation functionality $\mathcal{F}_{\mathsf{dkg}\mathcal{Q}}^{n,t}$ for $\mathbb{G}$ running with parties $P_1, \ldots, P_n$, a quorum $\mathcal{Q} \subseteq [n]$ of $t+1$ online parties, and corrupted parties $\mathcal{I} \subseteq [n]$ with $|\mathcal{I}| \leq t$ is defined as follows:

1. Wait to receive $(\mathsf{gen}, \mathsf{nonce}, \mathcal{Q})$ from all $t+1$ parties
2. If $(\mathsf{gen}, \mathsf{nonce}, \mathcal{Q}, p(x))$ has already been stored
   - Retrieve $p(x)$
   - Store $(\mathsf{gen}, \mathsf{nonce}, \mathcal{Q}, p(x))$

   Else
   - Choose a random degree-$t$ polynomial $p(x) \leftarrow\!\!\$\ \mathbb{F}_q[x]$
   - Store $(\mathsf{gen}, \mathsf{nonce}, \mathcal{Q}, p(x))$
3. For $j = 1, \ldots, n$, compute $k_j = p(\alpha_j)$
4. Let $a_0, \ldots, a_t$ be the coefficients of $p$; i.e., $p(x) = \sum_{i=0}^{t} a_i \cdot x^i$
5. For $i = 0, \ldots, t$, compute $A_i = a_i \cdot G$
6. Compute $k = p(0)$ and $K = k \cdot G$
7. For $i = 1, \ldots, n$, send $(\mathsf{gen}, \mathsf{nonce}, \mathcal{Q}, k_i, A_0, \ldots, A_t, K)$ to $P_i$

Observe that in the case of additive DKG, the functionality allowed each corrupted party to choose its own share (and the honest parties' shares were randomly chosen by the functionality). In contrast, here the functionality chooses all of the shares. The reason for this difference is that here each party's share is the sum of the shares received from all parties. Since each party is a priori committed to its values after the initialization phase, this means that no party can influence even its own share and so there is no need to give this extra power to the adversary.

**Protocol 5 ($\Pi_{\mathsf{dkg}\mathcal{Q}}^{n,t}$ for threshold DKG)**

- ***Initialize (all $n$ parties):***
  1. *In parallel, each $P_i$ sends $(\mathsf{init}, i)$ to $\mathcal{F}_{\mathsf{eVRF}}$ (for $\mathcal{X}, \mathcal{Y}$)*
  2. *Wait to receive $(\mathsf{init}, j, \mathsf{sid}_j)$ from $\mathcal{F}_{\mathsf{eVRF}}$ for all $j \in [n]$*
- ***Generate(a quorum $\mathcal{Q}$ of $t+1$ parties):*** *upon input $(\mathsf{gen}, \mathsf{nonce}, \mathcal{Q})$ with a $\mathsf{nonce}$ and with $\mathcal{Q} \subseteq [n]$ of size $t+1$, each party $P_i$ with $i \in \mathcal{Q}$*
  1. Message (all parties in $\mathcal{Q}$): *Each party $P_i$ with $i \in \mathcal{Q}$,*
     (a) *For $\ell = 0, \ldots, t$, send $(\mathsf{eval}, i, \mathsf{sid}_i, \mathsf{nonce} \| \mathcal{Q} \| \ell)$ to $\mathcal{F}_{\mathsf{eVRF}}$ and receive back $(\mathsf{eval}, i, \mathsf{sid}_i, \mathsf{nonce} \| \mathcal{Q} \| \ell, a_i^\ell, A_i^\ell)$. (Note that the nonce used for the evaluation includes the identities of the $t$ participating parties.)*
     (b) *Let $p_i(x) = \sum_{\ell=0}^{t} a_i^\ell \cdot x^\ell$*
     (c) *Compute $k_{i \to j} = p_i(\alpha_j)$ for $j = 1, \ldots, n$*
     (d) *Send $k_{i \to j}$ to $P_j$ for $j = 1, \ldots, n$*
  2. Output (all $n$ parties):
     (a) *Wait to receive $\{(\mathsf{eval}, j, \mathsf{sid}_j, \mathsf{nonce} \| \mathcal{Q} \| \ell, A_j^\ell)\}_{\ell=0}^{t}$ from $\mathcal{F}_{\mathsf{eVRF}}$ for all parties $P_j$ with $j \in \mathcal{Q}$ (all with the correct $\mathsf{nonce}$ and $\mathcal{Q}$)*
     (b) *Wait to receive $k_{j \to i}$ for all $j \in \mathcal{Q}$*
     (c) *Verify that $k_{j \to i} \cdot G = \sum_{\ell=0}^{t} (\alpha_i)^\ell \cdot A_j^\ell$, for all $j \in \mathcal{Q}$. Abort if any equality does not hold.*

*(d) Compute* $k_i = \sum_{j \in \mathcal{Q}} k_{j \to i}$
*(e) Compute* $A_\ell = \sum_{j \in \mathcal{Q}} A_j^\ell$ *for* $\ell = 0, \ldots, t$
*(f) Compute* $K = A_0$
*(g) Output* $(\mathsf{gen}, \mathsf{nonce}, \mathcal{Q}, k_i, A_0, \ldots, A_t, K)$

**Theorem 6.** *Protocol 5 securely realizes* $\mathcal{F}_{\mathsf{dkg}\mathcal{Q}}^{n,t}$ *with perfect security with abort, in the presence of a static malicious adversary corrupting up to* $t$ *parties. Furthermore, after a single two-round initialization phase, each generation consists of a single round only.*

*Proof.* The idea behind the proof is that the simulator can generate all of the corrupted parties' values itself (using $f$ obtained in the initialization phase of $\mathcal{F}_{\mathsf{eVRF}}$). Then, for all but one honest party, the simulator chooses their values at random. Finally, for a single specified honest party, its polynomial (in the exponent; i.e., it's $A_\ell^j$ values) are computed by subtracting all the other parties' polynomials from the polynomial received from $\mathcal{F}_{\mathsf{dkg}\mathcal{Q}}^{n,t}$. This ensures that all the polynomials of the parties add up to the random polynomial sent by $\mathcal{F}_{\mathsf{dkg}\mathcal{Q}}^{n,t}$. This is identical to a real execution in the $\mathcal{F}_{\mathsf{eVRF}}$-hybrid model since only "public values" are revealed.

We now proceed with the proof. If no parties are corrupted, then the statement is trivial. Let $\mathcal{A}$ be the adversary and let $\mathcal{I}$ be the set of corrupted parties with $0 < |\mathcal{I}| \leq t$. We construct a simulator $\mathcal{S}$ with "internal communication" to $\mathcal{A}$ and "external communication" to $\mathcal{F}_{\mathsf{dkg}\mathcal{Q}}^{n,t}$, as follows:

- *Initialize (all n parties):*[6]
  1. Simulate $\mathcal{F}_{\mathsf{eVRF}}$ sending $\{(\mathsf{init}, j)\}_{j \notin \mathcal{I}}$ to $\mathcal{A}$, and receive back $\{\mathsf{sid}_j\}_{j \notin \mathcal{I}}$ as $\mathcal{A}$ would send to $\mathcal{F}_{\mathsf{eVRF}}$
  2. Simulate $\mathcal{F}_{\mathsf{eVRF}}$ sending $(\mathsf{init}, j, \mathsf{sid}_j)$ to all parties, for all $j \notin \mathcal{I}$
  3. For all $i \in \mathcal{I}$, internally receive $(\mathsf{init}, i, \mathsf{sid}_i, f_i)$ from $\mathcal{A}$ as sent to $\mathcal{F}_{\mathsf{eVRF}}$ from $P_i$, and simulate $\mathcal{F}_{\mathsf{eVRF}}$ sending back $(\mathsf{init}, i, \mathsf{sid}_i)$ to all parties
- *Generate (a quorum* $\mathcal{Q}$ *of* $t+1$ *parties):* to simulate an execution of generate for $\mathsf{nonce}$ with quorum $\mathcal{Q}$,
  1. Send $(\mathsf{gen}, \mathsf{nonce}, \mathcal{Q})$ to $\mathcal{F}_{\mathsf{dkg}\mathcal{Q}}^{n,t}$ for every $i \in \mathcal{I} \cap \mathcal{Q}$, and receive back $\{(\mathsf{gen}, \mathsf{nonce}, \mathcal{Q}, k_i, A_0, \ldots, A_t, K)\}_{i \in \mathcal{I}}$
  2. For every $i \in \mathcal{I} \cap \mathcal{Q}$, compute $a_i^\ell = f_i(\mathsf{nonce} \| \mathcal{Q} \| \ell)$ for $\ell = 0, \ldots, t$, where $f_i$ is as received for $P_i$ in the initialize phase, and set $p_i(x) = \sum_{\ell=0}^t a_i^\ell \cdot x^\ell$
  3. For every $i \in \mathcal{I} \cap \mathcal{Q}$, compute $A_i^\ell = a_i^\ell \cdot G$
  4. For every $i \in \mathcal{I} \cap \mathcal{Q}$ and every $j \in \mathcal{Q} \setminus \mathcal{I}$, compute $k_{i \to j} = p_i(\alpha_j)$
  5. Let $j' \in \mathcal{Q} \setminus \mathcal{I}$ be a specific honest party (since $|\mathcal{I}| \leq t$ there exists such a party)
  6. For all $j \in \mathcal{Q} \setminus \mathcal{I}$ with $j \neq j'$, choose a random $p_j(x) = \sum_{\ell=0}^t a_j^\ell \cdot x^\ell$ and compute $A_j^\ell = a_j^\ell \cdot G$ for $\ell = 0, \ldots, t$
  7. For all $j \in \mathcal{Q} \setminus \mathcal{I}$ with $j \neq j'$, and for all $i \in \mathcal{I}$, compute $k_{j \to i} = p_j(\alpha_i)$
  8. For every $i \in \mathcal{I}$, compute $k_{j' \to i} = k_i - \sum_{j \in \mathcal{Q} \setminus \{j'\}} k_{j \to i}$, where $k_i$ is as received from $\mathcal{F}_{\mathsf{dkg}}^{n,t}$ (this ensures that for every corrupted party $P_i$ it holds that $\sum_{j \in \mathcal{Q}} k_{j \to i} = k_i$)
  9. For $\ell = 0, \ldots, t$, compute $A_{j'}^\ell = A_\ell - \sum_{j \in \mathcal{Q} \setminus \{j'\}} A_\ell^j$, where $A_0, \ldots, A_t$ are as received from $\mathcal{F}_{\mathsf{dkg}}^{n,t}$
  10. Simulate $\mathcal{F}_{\mathsf{eVRF}}$ sending $(\mathsf{eval}, j, \mathsf{sid}_j, \mathsf{nonce} \| \mathcal{Q} \| \ell, A_j^\ell)$ to all corrupted parties, for every $j \in \mathcal{Q} \setminus \mathcal{I}$ and $\ell = 0, \ldots, t$

[6] The simulation of this phase is exactly as in the proof of Theorem 4.

11. For each $j \in \mathcal{Q} \setminus \mathcal{I}$ and $i \in \mathcal{I}$, simulate honest $P_j$ sending $k_{j\to i}$ as computed above to corrupted $P_i$
12. Define the set $\mathcal{O}_{\mathsf{out}}$ of honest parties to receive output by the set of parties for which $\mathcal{A}$ instructs $\mathcal{F}_{\mathsf{eVRF}}$ to provide output from all $i \in \mathcal{I}$, and for which all $k_{i\to j}$ values are sent from corrupted parties to honest parties, and they are all valid ($\mathcal{S}$ can check validity as in the protocol)
13. Send $\mathcal{F}^{n,t}_{\mathsf{dkg}\mathcal{Q}}$ the set $\mathcal{O}_{\mathsf{out}}$
14. Output whatever $\mathcal{A}$ outputs

First observe that **(a)** for every $i \in \mathcal{I}$ it holds that $\sum_{j\in\mathcal{Q}} k_{j\to i} = k_i$, and **(b)** for every $j \in \mathcal{Q} \setminus \mathcal{I}$ and $i \in \mathcal{I}$ it holds that $k_{j\to i} \cdot G = \sum_{\ell=0}^{t} (\alpha_i)^\ell \cdot A^\ell_j$. The former follows immediately from how $k_{j'\to i}$ is chosen. Regarding the latter, this also holds trivially for all $j \neq j'$. Regarding $j'$, observe that $k_{j'\to i}$ is defined by $k_i - \sum_{j\in\mathcal{I}\setminus\{j'\}} k_{j\to i}$ and each $A^\ell_{j'}$ is defined by $A^\ell_{j'} = A_\ell - \sum_{j\in\mathcal{Q}\setminus\{j'\}} A^j_\ell$. Now, for all $k_{j\to i}$ with $j \neq j'$ we have that $k_{j\to i} \cdot G = \sum_{\ell=0}^{t} (\alpha_i)^\ell \cdot A^\ell_j$ and by the $\mathcal{F}^{n,t}_{\mathsf{dkg}}$ functionality definition we have that $k_i \cdot G = \sum_{\ell=0}^{t} (\alpha_i)^\ell \cdot A_\ell$. It therefore follows that

$$
\begin{aligned}
k_{j'\to i} \cdot G &= \left( k_i - \sum_{j\in\mathcal{Q}\setminus\{j'\}} k_{j\to i} \right) \cdot G = \sum_{\ell=0}^{t} (\alpha_i)^\ell \cdot A_\ell - \sum_{j\in\mathcal{Q}\setminus\{j'\}} \sum_{\ell=0}^{t} (\alpha_i)^\ell \cdot A^\ell_j \\
&= \sum_{\ell=0}^{t} (\alpha_i)^\ell \cdot \left( A_\ell - \sum_{j\in\mathcal{Q}\setminus\{j'\}} A^\ell_j \right) = \sum_{\ell=0}^{t} (\alpha_i)^\ell \cdot A^\ell_{j'},
\end{aligned}
$$

as required. Furthermore, since $\mathcal{F}^{n,t}_{\mathsf{dkg}\mathcal{Q}}$ chooses the polynomial to be random, and so do the honest parties, the distribution over these values is identical in the real and ideal execution. Finally, the simulation of the initialization phase is computationally indistinguishable from a real execution (as shown in Theorem 4) with the only difference being if there is a collision in the $\mathsf{sid}$, and the simulation of the generate phase yields a distribution that is identical to the protocol (in the $\mathcal{F}_{\mathsf{eVRF}}$ model) since the only thing that the corrupted parties sees are the $A^j_\ell$ values, and they are committed to their $A^j_i$ and $k_{i\to j}$ values by $\mathcal{F}_{\mathsf{eVRF}}$.

The round complexity statement in the theorem follows by observation that each $\mathcal{F}_{\mathsf{eVRF}}$ initialization operation is two rounds and each evaluation operation is just a single round (as described in Protocol 1, and these can all be sent in parallel. This completes the proof. □

**On knowing the set of generating parties $\mathcal{Q}$:** Protocol $\Pi^{n,t}_{\mathsf{gen}}$ assumes that all parties know (and agree upon) the set of $t+1$ participants $\mathcal{Q}$ ahead of time. This is needed to ensure that an independent key is generated for each nonce and subset, which is needed since each party has a different eVRF, and so different subsets would generate different keys, even for the same nonce. The set $\mathcal{Q}$ is included to therefore ensure complete independence of keys generated with different subsets. (If we only include the nonce, then it is possible that two different keys will be generated for which the adversary knows the "difference" between them; e.g., consider a case of $t$ honest parties running the execution twice on the same nonce, each time with a different corrupted party who is the $(t+1)$th party.) This limitation can be removed by simply having *all* parties participate in key generation, or by having a *fixed* set of parties who generate keys, or by adding an additional round to agree on the set of parties.

**One-round threshold DKG without knowing $\mathcal{Q}$ ahead of time:** As we have discussed, the DKG of Section 4.2 requires the parties to know the set $\mathcal{Q}$ ahead of time. In practice, this is not always a given, and the desired functionality is that after the first $t+1$ parties respond, the key is generated.[7] This can be achieved by using a threshold eVRF, which is an eVRF for which any authorized quorum of parties $\mathcal{Q}$ can compute the output $Y$ and receive a sharing of $y$ (where $Y = y \cdot G$). This solves the problem of knowing $\mathcal{Q}$ ahead of time since any $t+1$ parties can participate in computing the eVRF output, which can be used directly as the generated key. We leave the construction of a threshold eVRF as an open question (see Section 7). However, for a (very) small $n$, it is possible to achieve the desired result by simply calling $\mathcal{F}_{\mathsf{dkg}\mathcal{Q}}^{n,t}$ separately for every authorized subset $\mathcal{Q} \subset [n]$ and taking the result using the $\mathcal{Q}$ that defines the set of $t+1$ parties who actually participated (i.e., sent first round messages). This is not efficient for a large number of parties, but can certainly be used for thresholds of the type 2-out-of-3 (requiring 3 computations) or 3-out-of-5 (requiring 10 computations).

### 4.3 The Transformation Methodology for Signing Protocols

ECDSA and Schnorr signing both involve generating a random nonce $k$ and revealing $R = k \cdot G$. In ECDSA, the signature is $(r, s)$, where $r$ is derived from the $x$-value of $R$ and $s = k^{-1} \cdot (H(m) + r \cdot x)$ with $x$ being the private key and $m$ the message to be signed. In Schnorr (using comparable notation), the signature is $(R, s)$ where $s = k - e \cdot x \bmod q$ (or some variant of this equation) and $e = H(Q, R, m)$, where $Q = x \cdot G$ is the public key.

Many protocols that achieve simulation – e.g., [26,16,15] for ECDSA and [27,28] for Schnorr – work by having each party choose a random $k_i$ and commit to $R_i = k_i \cdot G$ (sometimes also including a zero-knowledge proof of knowledge of the discrete log) and then having the parties decommit and define $R = \sum_{i=1}^{n} R_i$ as the nonce. This methodology ensures that $R$ is uniformly distributed (since no party can make their $R_i$ depend on the others due to the commitments). When extractable and equivocal commitments are used this is also fully simulatable, since a simulator can choose the honest parties' $R_i$ values after seeing the corrupted parties values, and can therefore make the sum equal the value $R$ received externally in the ideal model. *Stated differently, many protocols generate $R$ by running a simulatable distributed key generation protocol.* When looked at in this light, a natural transformation is to use our one-round DKG protocols (Protocols 3 and 5) to generate $R$ (applying the eVRF to the message to be signed and/or a unique nonce), with the initialization phase being run together with the signing key generation protocol. This enables us to collapse two rounds (commit and open) into a single round, thereby reducing the number of rounds in the signing protocol from three to two, while still achieving simulatable DKG. This enables us to reduce the number of rounds from three to two, without sacrificing on full simulatability. Furthermore, by applying the eVRF to the message to be signed only, we have the all signatures on the same message will have the same nonce, and so *deterministic* signing is achieved at no additional cost.

**On achieving a black-box transformation.** Ideally, we would like to prove a theorem that says something like any protocol that has a "coin tossing phase" where parties exchange $R_1, \ldots, R_n$ (e.g., by committing and opening) and the nonce is $R = \sum_{i=1}^{n} R_i$ and transform it into a deterministic protocol where the $R_i$ values are generated via the eVRF. However, this actually isn't possible when

[7] Consider the case of human participants who receive an "invite" to participate in a DKG, and who connect to run the operation and then disconnect. It is much easier to not have to know who the parties that join are ahead of time, and whoever the first $t+1$ parties are, the DKG will go through.

constructing deterministic signing. In order to see this, take *any* secure ECDSA or Schnorr protocol for probabilistic signing and modify it so that if any of the parties guesses in the first round what the nonce $R$ will be, then all parties send them their private key share. Such a protocol will completely break when deterministically generating $R$ from the message, like with our eVRF, when the same message is signed twice. This does not rule out the possibility of constructing probabilistic signing via a general transformation, but such a transformation is unlikely to be very useful since most existing protocols are not proven with the nonce generation as a separate modular operation. In the following, we therefore present specific protocols which are derived from taking existing protocols and replacing the commit-and-open phase with an eVRF computation.

### 4.4 Two-Round Simulatable Multiparty Schnorr Signing

In this section, we construct two-round multiparty Schnorr signing from the protocol by [27], by replacing the commit-and-open rounds by eVRF evaluations, as described above. We begin by constructing $n$-out-of-$n$ deterministic signing, and then describe how to achieve probabilistic signing and threshold signing.

**Deterministic signing with additive shares:** We first consider the case where the parties hold additive shares of the signing key, and all $n$-of-$n$ parties participate in signing. We begin by defining the signing functionality. This functionality computes the standard Schnorr signature for a set of parties with additively shared keys. The functionality does not mandate how the key is generated, and it works for any set of inputs held by the parties. This guarantees the same level of security as locally computed Schnorr no matter how keys are generated (using some HD scheme, poorly derived from passwords, or anything else).

**Functionality 7 (Deterministic Schnorr Signing $\mathcal{F}_{\mathsf{det\text{-}schnorr}}$)**

*Let $\mathbb{G}$ be a group of order $q$ with generator $G$, and let $H$ be the Schnorr hash function. Upon receiving $(\mathsf{Sign}, m, Q, Q_1, \ldots, Q_n, x_i)$ from all $n$ different parties $P_i$, functionality $\mathcal{F}_{\mathsf{det\text{-}schnorr}}$ works as follows:*

1. *Verifies that all parties sent the same $(m, Q, Q_1, \ldots, Q_n)$ , that $Q = \sum_{i=1}^{n} Q_i$ and that $Q_i = x_i \cdot G$ for all $i \in [n]$. If no, then it does nothing. Else, it proceeds to the next step.*
2. *Computes $x = \sum_{i=1}^{n} x_i \bmod q$.*
3. *If some $(m, k)$ is stored then retrieves $k$. Else, chooses a random $k \leftarrow\!\!\$\ \mathbb{Z}_q$ and stores $(m, k)$.*
4. *Computes $R = k \cdot G$, $e = H(Q \| R \| m)$ and $s = k - e \cdot x \bmod q$.*
5. *Sends $(m, e, s)$ to all parties.*

**Securely computing $\mathcal{F}_{\mathsf{det\text{-}schnorr}}$.** The idea behind the protocol for Schnorr is simple, due to the fact that the Schnorr signing equation is linear. Specifically, the parties use an eVRF to generate partial nonces $(k_i, R_i)$ where $R_i = k_i \cdot G$ and to share all $R_1, \ldots, R_n$ with all parties. Then, each party can locally compute $R = \sum_{i=1}^{n} R_i$, $e = H(Q \| R \| m)$ and $s_i = k_i - e \cdot x_i \bmod q$. This implies that $\sum_{i=1}^{n} s_i = k - e \cdot x \bmod q$, and so $(e, s)$ is a valid signature. The fact that the signing is deterministic is achieved by applying the eVRF to the (hash of the) message as input. Essentially, this is exactly the same as for the standard EdDSA signing scheme [2], except that a different pseudorandom function is used.

**Protocol 8 (Multiparty Schnorr signing – $n$-out-of-$n$ parties)**

– **Input:**
  1. Group parameters: *Let $\mathbb{G}$ be a group of order $q$ with generator $G$, and let $H$ be the Schnorr hash function with output length $\lambda'$*
  2. Key shares: *Each party $P_i$ holds $(m, x_i, Q, Q_1, \ldots, Q_n)$ where $Q_i = x_i \cdot G$ and $\sum_{i=1}^{n} Q_i = Q$*
  3. eVRF shares: *Each party $P_i$ holds $(\mathsf{sk}_i, \mathsf{vk}_1, \ldots, \mathsf{vk}_n)$ where $\mathsf{sk}_i$ is the eVRF private key associated with $\mathsf{vk}_i$, for an eVRF with domain $\{0,1\}^{\lambda'}$ and range $\mathbb{G}$. These are generated in parallel using Protocol 1 ($\pi_{EF}$).*
– **The protocol:**
  1. Round 1: *Each party $P_i$ computes $(k_i, R_i, \pi_i) \leftarrow \mathsf{Eval}(\mathsf{sk}_i, H(m))$, and sends $(R_i, \pi_i)$ to all parties*
  2. Round 2: *Upon receiving $(R_j, \pi_j)$ from all parties $j \in [n]$, each $P_i$:*
     (a) *Proceeds if $\mathsf{Verify}(\mathsf{vk}_j, H(m), R_j, \pi_j) = 1$ for all $j \in [n]$ and aborts otherwise*
     (b) *Computes $R = \sum_{j=1}^{n} R_j$, $e = H(Q \| R \| m)$ and $s_i = k_i - x_i \cdot e \bmod q$*
     (c) *Sends $s_i$ to all parties*
  3. Output: *Upon receiving $(s_1, \ldots, s_n)$, each party computes $s = \sum_{i \in S} s_i \bmod q$ and checks that $\mathsf{Verify}_Q(m, (s, e)) = 1$. If yes, then it outputs $(s, e)$; otherwise it aborts.*

Our protocol assumes that the parties hold **valid and consistent inputs**, meaning that all parties hold the same vectors $(Q, Q_1, \ldots, Q_n)$ and $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$, and in addition $Q = \sum_{i=1}^{n} Q_i$, and $Q_i = x_i \cdot G$ and $\mathsf{vk}_i = \mathsf{sk}_i \cdot G$ for every $i \in [n]$. If this is not guaranteed from a previous protocol execution, then each $P_i$ can simply verify that $x_i \cdot G = Q_i$ and $Q = \sum_{i=1}^{n} Q_i$ at the beginning of the protocol. In addition, all parties can send $h_q \leftarrow H(Q_1, \ldots, Q_n, \mathsf{vk}_1, \ldots, \mathsf{vk}_n)$ to all other parties in the first round, and proceed in the second round only if the same hash value $h_q$ is received from all.

**The protocol for two parties.** In the specific case of two parties, and where only one party needs to receive output, Protocol 8 can be converted into a protocol where $P_1$ sends a single message to $P_2$, and $P_2$ replies with a single message to $P_1$ (i.e., a single round trip), and $P_1$ can then generate output.

**Security.** We prove the protocol secure in the $\mathcal{F}_{\mathsf{eVRF}}$-hybrid model (assuming that the initialize phase is carried out during key generation). We stress that the "perfect security" in the theorem statement only holds in the $\mathcal{F}_{\mathsf{eVRF}}$-hybrid model, but when instantiating the protocol with a real eVRF, the protocol is computationally secure. In addition, we remark that security holds for *all* valid and consistent inputs $\{(x_i, Q, Q_1, \ldots, Q_n)\}_{i \in [n]}$ irrespective of how they are generated. In contrast, it is crucial that the eVRF inputs are securely generated; this is reflected in the proof by the fact that we consider the $\mathcal{F}_{\mathsf{eVRF}}$-hybrid model.

**Theorem 9.** *Assume that the parties hold valid and consistent inputs. Then, Protocol 8 securely computes functionality $\mathcal{F}_{\mathsf{det\text{-}schnorr}}$ in the $\mathcal{F}_{\mathsf{eVRF}}$-hybrid model with perfect security with abort, in the presence of a malicious static adversary controlling any subset of the parties.*

*Proof.* The idea behind the proof of security is simple. Using the simulatability of the eVRF (as demonstrated in the proof of Theorem 4) the simulator can force the sum of all $R_i$'s to equal the $R$ value it receives in the signature from the ideal functionality $\mathcal{F}_{\mathsf{det\text{-}schnorr}}$. Then, the values $s_i$ sent by the honest parties in the protocol can be derived perfectly by choosing the honest $R_j$ values carefully for all but one honest party, and using the signature for the last honest party, exactly as in [27].

Let $\mathcal{A}$ be an adversary corrupting a (strict) subset of parties $I \subset [n]$ of size at most $n-1$ (if all $n$ are corrupted, then the protocol is vacuously secure), and let $J$ denote the set of honest parties (and so $I \cup J = [n]$). Without loss of generality, assume that $1 \in J$ (i.e., $P_1$ is an honest participant). We are now ready to construct the simulator $\mathcal{S}$, with input $\{(\mathsf{Sign}, m, Q, Q_1, \ldots, Q_n, x_i)\}_{i \in I}$, as follows:

1. $\mathcal{S}$ externally sends $(\mathsf{Sign}, m, Q, Q_1, \ldots, Q_n, x_i)$ to $\mathcal{F}_{\mathsf{det\text{-}schnorr}}$ and receives back $(m, e, s)$. $\mathcal{S}$ computes $R = s \cdot G + e \cdot Q$. Then, $\mathcal{S}$ invokes $\mathcal{A}$ in an execution of the protocol.
2. Let $f_i$ be the stored function from the $\mathcal{F}_{\mathsf{eVRF}}$ initialization phase for party $P_i$, for every $i \in [n]$ (as in the proof of Theorem 4, the simulator $\mathcal{S}$ has these functions), and let $\mathsf{sid}$ be the identifier.
3. For all $i \in I$, simulator $\mathcal{S}$ computes $k_i = f_i(H(m))$ and $R_i = k_i \cdot G$.
4. For all $j \in J \setminus \{1\}$, simulator $\mathcal{S}$ chooses a random $s_j \leftarrow \mathbb{Z}_q$ and sets $R_j = s_j \cdot G + e \cdot Q_j$ (where $e$ is from the signature received from $\mathcal{F}_{\mathsf{schnorr}}$). Then, $\mathcal{S}$ sets $R_1 = R - \sum_{i \in I} R_i - \sum_{j \in J \setminus \{1\}} R_j$, using $R$ computed from the signature received from $\mathcal{F}_{\mathsf{det\text{-}schnorr}}$.
5. $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{eVRF}}$ sending $(\mathsf{eval}, \mathsf{sid}, j, H(m), R_j)$ to $\mathcal{A}$ for every $j \in J$, using the $R_j$ values computed in the previous step.
6. $\mathcal{S}$ receives $(\mathsf{eval}, \mathsf{sid}, i, H(m))$ from $\mathcal{A}$ as sent to $\mathcal{F}_{\mathsf{eVRF}}$ for every $i \in I$. $\mathcal{S}$ waits for all messages to be sent.
7. $\mathcal{S}$ computes $s_i = k_i - x_i \cdot e \bmod q$ for every $i \in I$ ($\mathcal{S}$ can do this since it knows the $k_i$ values for each corrupted party from Step 3 above, and it is given the $x_i$ values of the corrupted parties as input). Then, $\mathcal{S}$ computes

$$s_1 = s - \sum_{i \in I} s_i - \sum_{j \in J \setminus \{1\}} s_j \pmod q$$

   using the $s_j$ values chosen above.
8. $\mathcal{S}$ simulates $P_j$ sending $s_j$ to all parties, for every $j \in J$.
9. $\mathcal{S}$ receives $\{s_i\}_{i \in I}$ values sent by $\mathcal{A}$ to the honest parties. If the sum of all of the values sent to an honest $P_j$ is correct (computed by $\sum_{i \in I} s_i$ where the $s_i$ values are as above), then $\mathcal{S}$ adds $P_j$ to $\mathcal{O}_{\mathsf{out}}$.
10. $\mathcal{S}$ sends $\mathcal{O}_{\mathsf{out}}$ to $\mathcal{F}_{\mathsf{det\text{-}schnorr}}$ to instruct which honest parties should receive output.

This completes the simulation. We argue that the simulation is *perfect*. In order to see this, we show that the $(R_j, s_j)$ values sent by the simulator to the adversary are identically distributed to the values sent by the honest parties to the corrupted parties in a real protocol execution. In order to see this, first note that for every $j \in J \setminus \{1\}$ the values are generated as follows:

- *Real:* $k_j \in_R \mathbb{Z}_q$ is random, $R_j = k_j \cdot G$, and $s_j = k_j - e \cdot x_j \bmod q$
- *Simulation:* $\tilde{s}_j \in_R \mathbb{Z}_q$ is random, $\tilde{R}_j = \tilde{s}_j \cdot G + e \cdot Q_j$ (we write $\tilde{s}_j$ and $\tilde{R}_j$ to differentiate from the real)

Let $\tilde{k}_j$ be such that $\tilde{R}_j = \tilde{k}_j \cdot G$. We remark that the simulator $\mathcal{S}$ does not know $\tilde{k}_j$, but the value is well defined. It follows that $\tilde{k}_j = \tilde{s}_j + e \cdot x_j \bmod q$ and so $\tilde{s}_j = \tilde{k}_j - e \cdot x_j \bmod q$, exactly like in a real execution. Furthermore, choosing $\tilde{k}_j$ at random and computing $\tilde{s}_j = \tilde{k}_j - e \cdot x_j \bmod q$ yields the exact same distribution as choosing $\tilde{s}_j$ at random and computing $\tilde{k}_j = \tilde{s}_j + e \cdot x_j \bmod q$.

Next, regarding $(R_1, s_1)$, we have that

$$R_1 = R - \sum_{i \in I} R_i - \sum_{j \in J \setminus \{1\}} R_j = k \cdot G - \sum_{i \in I} k_i \cdot G - \sum_{j \in J \setminus \{1\}} k_j \cdot G$$

where $k$ is the discrete log of $R$ (as computed from the signature), $\{k_i\}_{i\in I}$ are the corrupted parties' values from the eVRF (enforced by the $f_i$ functions), and $\{k_j\}_{j\in J\setminus\{1\}}$ are the implicit values defined above. This therefore defines $k_1 = k - \sum_{i\in I} k_i - \sum_{j\in J\setminus\{1\}} k_j \bmod q$. Similarly, we have

$$s_1 = s - \sum_{i\in I} s_i - \sum_{j\in J\setminus\{1\}} s_j = k - e\cdot x - \sum_{i\in I}(k_i - e\cdot x_i) - \sum_{j\in J\setminus\{1\}}(k_j - e\cdot x_j) \pmod q$$

which holds for $i \in I$ by how the simulator computes $\{s_i\}_{i\in I}$ and for $j \in J \setminus \{1\}$ by the above analysis. Writing $k = \sum_{\ell\in I\cup J} k_\ell$ and $d = \sum_{\ell\in I\cup J} x_\ell$ we have that

$$s_1 = \sum_{\ell\in I\cup J} k_\ell - e\cdot \sum_{i\in I\cup J} x_\ell - \sum_{i\in I}(k_i - e\cdot x_i) - \sum_{j\in J\setminus\{1\}}(k_j - e\cdot x_j) \pmod q$$

and so $s_1 = k_1 - e\cdot x_1 \bmod q$, as required. (We stress that $\mathcal{S}$ does not know these values, and in particular it does not know the $x_j$ values of the honest parties including $x_1$, and yet is able to generate the correct distribution, as described above.)

Finally, since $\mathcal{S}$ is able to perfectly verify whether or not the corrupted parties send correct values, since it knows all of the corrupted $(k_j, x_j)$ values and so can detect if the *sum* over all $s_i$ values sent by $\mathcal{A}$ is correct. (Note that only the sum matters for $\mathcal{C}$ computing a correct signature.) Thus, the distribution over $\mathcal{C}$ receiving or not receiving output is exactly the same in the real and ideal executions. □

**Security under concurrent composition.** As shown by [25], perfect security without rewinding implies UC security. As such, assuming that $\mathcal{F}_{\mathsf{eVRF}}$ is implemented using a UC-secure protocol (as in Protocol 1), we have that the protocol is UC-secure and so secure under concurrent composition.

**The final result.** Using any two-round distributed key generation protocol (e.g., as described in [27]) in parallel with the two-round initialization in Protocol 1, we have the following corollary:

**Corollary 2.** *There exists a multiparty n-of-n protocol with two rounds for each of key generation and signing that UC computes the deterministic signing functionality $\mathcal{F}_{\mathsf{det\text{-}schnorr}}$ with abort, in the presence of a malicious static adversary controlling any subset of the parties.*

**Probabilistic signing.** We can achieve two-round probabilistic signing assuming that the parties hold the same *unique nonce*[8] before the protocol begins by having the parties apply the eVRF to $H(H(m), \mathsf{nonce})$. This will result in the eVRF output being (computationally) independent for every different nonce. Practically, the nonce can be a timestamp, with the observation that if two protocol executions use the same timestamp, then the result will just be the same and so no harm can be done. Formally, the functionality computed here $\mathcal{F}_{\mathsf{schnorr}}$ would either choose $k \leftarrow\!\$\, \mathbb{Z}_q$ randomly each time (when a unique nonce is guaranteed) or would receive a nonce from all parties in the input and would choose a new $k \leftarrow\!\$\, \mathbb{Z}_q$ for every unique nonce (in which case, the functionality works in the same way that the same nonce in different execution would yield the same result, while different nonces would yield a different random $R$ in the signing).

**Threshold probabilistic signing.** In order to achieve probabilistic threshold signing, the parties can include the set of participating parties $\mathcal{Q}$ into the eVRF computation, together with $H(m)$ and

[8] Not to be confused with the "nonce" $R$ in the Schnorr signing, here we mean a unique value $\mathsf{nonce}$ which can be a counter, timestamp, or anything.

the $\mathsf{nonce}$, in the same way as in Protocol 5 for securely computing $\mathcal{F}^{n,t}_{\mathsf{dkg}\mathcal{Q}}$. As long as the set of participating parties is known ahead of time (since $\mathcal{Q}$ needs to be input into the eVRF evaluation), this achieves a two-round protocol. In the same way as for $\mathcal{F}^{n,t}_{\mathsf{dkg}\mathcal{Q}}$, if $n$ is very small then it is possible for the parties to provide eVRF values for all possible $\mathcal{Q}$ subsets, and so the set of parties need not be known ahead of time. However, this is only practical for very small $n$.

**Proof of quorum identity.** The threshold probabilistic signing protocol described above (that works by including $\mathcal{Q}$ into the eVRF) has a unique property that is of independent interest. The signature generated by the quorum of parties is a *standard signature* with no changes at all. However, the quorum of parties who signed can at a later time provide a proof that *they and only they* generated the signature, assuming a public record of the eVRF verifications keys. This proof for a signature $(e, s)$ is simply the set $\{(R_i, \pi_i\}_{i\in\mathcal{Q}}$, and it is verified by checking that $\mathsf{Verify}(\mathsf{vk}_i, H(m), R_i, \pi_i) = 1$ for all $i \in \mathcal{Q}$ and that $\sum_{i\in\mathcal{Q}} R_i = R$, where $R = s \cdot G + e \cdot Q$. The fact that the proof is sound (i.e., it isn't possible to frame another subset of parties) follows from the fact that if a party $P_j$ did not compute $\mathsf{Eval}$ on this input then their $(R_j, \pi_j)$ value is unknown, and furthermore even if they did (at some later time) the probability that the sum of $R_j$'s for any given subset $\mathcal{Q}'$ will equal a given $R$ is $1/q$ (in the $\mathcal{F}_{\mathsf{eVRF}}$-hybrid model).[9] Consider for example a setting, where like in a proof of stake, there is a reward for generating a signature and a penalty (slash) for generating a signature when you shouldn't. In such a setting, a proof of quorum identity provides a perfect solution. Furthermore, our protocol can enhance privacy by not revealing the identities of who signed except when needed, or except to entities who need to see it.

**Precomputation for the nonce.** Functionally speaking, there is no problem for the parties to precompute the nonce $R$ before the message is known (of course, this makes sense only for probabilistic signing). However, security wise, such a protocol would not securely realize $\mathcal{F}_{\mathsf{schnorr}}$ since the functionality provides a full signature $(e, s)$ – which fully determines $R = s \cdot G + e \cdot Q$ – only upon receiving $m$. Thus, one could define a different functionality $\mathcal{F}'_{\mathsf{schnorr}}$ that can be queried with $(\mathsf{precompute}, \mathsf{nonce})$ to obtain $R = k \cdot G$ (where $k$ is chosen randomly for each different $\mathsf{nonce}$), and then later when all parties send $(\mathsf{Sign}, m, R, Q, Q_1, \ldots, Q_n, x_i)$, the functionality will finalize the signature using the $R$ provided by all parties as input (and will refuse to sign on any other message with this $R$). It is straightforward to see that in this way, the protocol with precomputation securely realizes $\mathcal{F}'_{\mathsf{schnorr}}$. However, we stress that this requires parties to ensure that a new nonce is truly used in every precomputation (since the use of the same nonce in two precomputations would yield the same $R$ which would enable signing on two messages with the same $R$ which would leak the private key). Furthermore, the security of this functionality needs to be justified. That is, the Schnorr signing scheme needs to be proven from scratch in a model where $R$ can be given before $m$.

**Threshold deterministic signing.** As we have mentioned, we do not achieve deterministic signing for the threshold setting. This is because different subsets of parties compute a different $R$ value. In order to achieve this, we need to construct a threshold eVRF, which is left as an open question.

[9] If $t, n$ are very large, and so $\binom{t}{n}$ is not much smaller than $q$, then given all $\{(R_j, \pi_j)\}_{j\in[n]}$'s it may be possible to find an appropriate subset (this would require solving a type of subset sum problem, which may or may not be hard). However, if $\binom{t}{n} \ll q$, then the probability that there exists any such subset is negligible.

### 4.5 Two-Round Simulatable Two-Party ECDSA Signing

In a similar way to Section 4.4, in this section we construct a two-round two-party ECDSA signing protocol by replacing the commit-and-open phase in the protocol of [26] with an eVRF evaluation. The result is a protocol with a single round from $P_1$ to $P_2$, and a single response from $P_2$ to $P_1$.

**Functionality 10 (Two-party deterministic ECDSA Signing $\mathcal{F}_{\mathsf{det\text{-}ecdsa}}$)** *Let $\mathbb{G}$ be a group of order $q$ with generator $G$, and let $H$ be the ECDSA hash function. Upon receiving $(\mathsf{Sign}, m, Q, x_i)$ from both parties $P_1$ and $P_2$, functionality $\mathcal{F}_{\mathsf{det\text{-}ecdsa}}$ works as follows:*

1. *Verifies that both parties sent the same $(m, Q)$ , and that $(x_1 + x_2) \cdot G = Q$. If no, then it does nothing. Else, it proceeds to the next step.*
2. *Computes $x = x_1 + x_2 \bmod q$.*
3. *If some $(m, k)$ is stored then retrieves $k$. Else, chooses a random $k \leftarrow\!\!\$\ \mathbb{Z}_q$ and stores $(m, k)$.*
4. *Computes $R = k \cdot G$ and $r = r_x \bmod q$ where $R = (r_x, r_y)$*
5. *Computes $s = k^{-1} \cdot (H(m) + r \cdot x) \bmod q$.*
6. *Sends $(m, r, s)$ to party $P_1$ and to the ideal adversary $\mathcal{S}$.*[10]

We have defined the functionality so that only $P_1$ receives output. It is always possible to have $P_1$ send $P_2$ the output, if both are supposed to received the signature.

**The protocol idea and differences from [26].** The protocol of [26] uses the Paillier additively homomorphic encryption scheme to enable the parties to generate a signature. In the key generation phase, the parties obtain $x_1, x_2$ such that $x = x_1 + x_2 \bmod q$ (although [26] refers to multiplicative sharing, the protocol works for additive sharing in the same way). In addition, $P_1$ generates a Paillier key, and sends an encryption $c_{\mathsf{key}}$ of $x_1$ to $P_2$, together with a proof that $c_{\mathsf{key}}$ is correctly formed.

Next in order to sign, the parties first generate $R = k_1 \cdot k_2 \cdot G$, where $k_i$ is known to party $P_i$. Then, given the encryption $c_{\mathsf{key}}$ of $x_1$ and given $R$ (and thus $r$), it is possible for $P_2$ to generate an encryption of an "almost" signature. In particular, it can compute an encryption of $k_2^{-1} \cdot (H(m) + r \cdot x)$ by adding $x_2$ to $x_1$ inside $c_{\mathsf{key}}$, and then multiplying the result by $r$, adding $H(m)$, and finally multiplying again by $k_2^{-1}$ (since the operations inside Pailier are over the integers, it also adds $\rho \cdot q$ for a random $\rho$ of the appropriate size). Finally, given this ciphertext, party $P_1$ can decrypt and multiply the result by $k_1^{-1}$, yielding a "full" signature $k^{-1} \cdot (H(m) + r \cdot x)$.

The main difference between our protocol here and that of [26] is that we generate $k_1$ and $k_2$ (for $R = k_1 \cdot k_2 \cdot G$) using the eVRF. In this way, instead of doing commit-and-open, we are able to reduce the protocol to two messages (a single message in each direction), like Protocol 8 for Schnorr signatures. In addition, the protocol of [26] achieves only a game-based definition of security under standard assumptions, and requires an ad-hoc assumption regarding Paillier to achieve simulation-based security. In addition, it requires $P_1$ to refuse to run additional executions with $P_2$ if it is caught cheating in the last message. This limits the ability to run concurrent independent executions with the same key, making some deployment scenarios difficult. (As pointed out in [29], this is necessary since it is possible to actually extract the key one bit at a time if executions are not halted upon cheating.) In order to avoid the requirement to halt if someone attempts to cheat, we add a zero-knowledge proof from $P_2$ to $P_1$ (as recommended in [29]) that the ciphertext $c_{\mathsf{key}}$ is correctly

[10] We need the functionality to provide the signature to $\mathcal{S}$ for the case that $P_2$ is corrupted since $R$ is revealed to $P_2$ during the protocol execution, but only $P_1$ receives the signature for output. Thus, we give the signature to the adversary as well (in any case) in order to simulate.

computed. We have implemented this proof, and it takes 17ms to compute and 11ms to verify (on a 2019 MacBook Pro with a 2.3 GHz 8-Core Intel Core i9 processor). This adds to the running time but is not a problem for most applications.

**Protocol 11 (Two-party ECDSA signing)**

- **Input:**
  1. Group parameters: *Let $\mathbb{G}$ be a group of order $q$ with generator $G$, and let $H$ be the ECDSA hash function with output length $\lambda'$*
  2. Key shares: *Each party $P_i$ holds $(x_i, Q)$. In addition, $P_1$ holds a Paillier key $(N, \phi(N))$, and $P_2$ holds $c_{\mathsf{key}} = \mathsf{Paillier\text{-}enc}_N(x_1)$; these are generated exactly as in [26]*
  3. eVRF shares: *Each party $P_i$ holds $(\mathsf{sk}_i, \mathsf{vk}_1, \mathsf{vk}_2)$ where $\mathsf{sk}_i$ is the eVRF private key associated with $\mathsf{vk}_i$, for an eVRF with domain $\{0,1\}^{\lambda'}$ and range $\mathbb{G}$. These are generated in parallel using Protocol 1 ($\pi_{EF}$).*
- **The protocol:**
  1. Round 1 – $P_1$ to $P_2$: *Party $P_i$ computes $(k_1, R_1, \pi_1) \leftarrow \mathsf{Eval}(\mathsf{sk}_1, H(m))$, and sends $(R_1, \pi_1)$ to party $P_2$*
  2. Round 2 – $P_2$ to $P_1$: *Upon receiving $(R_1, \pi_1)$ from $P_1$, party $P_2$ works as follows,*
     (a) *Aborts if $\mathsf{Verify}(\mathsf{vk}_1, H(m), R_1, \pi_1) = 0$*
     (b) *Computes $(k_2, R_2, \pi_2) \leftarrow \mathsf{Eval}(\mathsf{sk}_2, H(m))$*
     (c) *Computes $R = k_2 \cdot R_1$ and $r = r_x \bmod q$ where $R = (r_x, r_y)$*
     (d) *Chooses a random $\rho \leftarrow \mathbb{Z}_{q^2}$ and random $\tilde{r} \in \mathbb{Z}_N^*$ (verifying explicitly that $\gcd(\tilde{r}, N) = 1$), and computes*

$$c = \mathsf{Paillier\text{-}enc}_N\left([k_2^{-1} \cdot H(m) \bmod q] + [k_2^{-1} \cdot r \bmod q] \cdot (x_2 + x_1) + \rho \cdot q\right)$$

     *using the Paillier homomorphic operations, including ciphertext rerandomization.*
     (e) *Sends a proof $\pi_c$ that $c$ is generated correctly, as in [29]*
     (f) *Sends $(R_2, \pi_2, c, \pi_c)$ to $P_1$*
  3. Output: *Upon receiving $(R_2, \pi_2, c, \pi_c)$, party $P_2$ works as follows,*
     (a) *Aborts if $\mathsf{Verify}(\mathsf{vk}_2, H(m), R_2, \pi_2) = 0$.*
     (b) *Aborts if $\pi_c$ is not an accepting proof that $c$ was generated correctly.*
     (c) *Computes $R = k_1 \cdot R_2$ and $r = r_x \bmod q$, where $R = (r_x, r_y)$.*
     (d) *Computes $s' = \mathsf{Paillier\text{-}dec}_{\phi(N)}(c)$ and $s'' = k_1^{-1} \cdot s' \bmod q$. (Also sets $s = \min\{s'', q - s''\}$ to ensures that the signature is always the smaller of the two possible values.)*
     (e) *Verifies that $(r, s)$ is a valid signature on $m$ with public key $Q$. If yes, outputs the signature $(r, s)$; otherwise, outputs $\mathsf{abort}$.*

**Security.** We now prove security of the protocol; the ideas are a combination of the proof of Theorem 9 (for the generation of $R$) together with the proof of the original protocol in [26].

**Theorem 12.** *Protocol 11 securely computes functionality $\mathcal{F}_{\mathsf{det\text{-}ecdsa}}$, in the presence of a malicious static adversary controlling any subset of the parties.*

*Proof.* We separately prove security for the case of a corrupted $P_1$ and a corrupted $P_2$. Let $\mathcal{A}$ be an adversary who has corrupted $P_1$; we construct a simulator $\mathcal{S}$. We prove only the signing protocol, as the key generation protocol is exactly as from [26] together with the eVRF key generation already proven in $\pi_{EF}$ (Protocol 1). We prove security in the $\mathcal{F}_{\mathsf{eVRF}}$-hybrid model, as in Theorem 9.

**Simulating signing – corrupted $P_1$:** The idea behind the security of the signing protocol is that a corrupted $P_1$ cannot do anything since all it does is participate in the generation of $R$ and then decrypts the ciphertext $c$ from $P_2$. Thus, the prove merely requires proving that a simulator can generate the corrupted $P_1$'s view of the decryption of $c$, given only the signature $(r, s)$ from $\mathcal{F}_{\mathsf{det\text{-}ecdsa}}$.

1. Upon input $(\mathsf{Sign}, m, Q, x_1)$, simulator $\mathcal{S}$ sends $(\mathsf{Sign}, m, Q, x_1)$ to $\mathcal{F}_{\mathsf{det\text{-}ecdsa}}$ and receives back a signature $(r, s)$ on the message $m$.
2. $\mathcal{S}$ computes the point $R$ from the signature $(r, s)$, using the ECDSA verification procedure.
3. Let $f_1$ be the stored function from the $\mathcal{F}_{\mathsf{eVRF}}$ initialization phase with identifier $\mathsf{sid}$ ($\mathcal{S}$ has this function and $\mathsf{sid}$).
4. $\mathcal{S}$ invokes $\mathcal{A}$ with input $(\mathsf{Sign}, m, Q, x_1)$ and receives $(\mathsf{eval}, \mathsf{sid}, 1, H(m))$ from $\mathcal{A}$ as sent to $\mathcal{F}_{\mathsf{eVRF}}$.
5. $\mathcal{S}$ computes $k_1 = f_1(H(m))$, $R_1 = k_1 \cdot G$, and $R_2 = k_1^{-1} \cdot R$.
6. $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{eVRF}}$ sending $(\mathsf{eval}, \mathsf{sid}, 2, H(m), R_2)$ to $\mathcal{A}$
7. $\mathcal{S}$ chooses a random $\rho \leftarrow \mathbb{Z}_{q^2}$, computes $c \leftarrow \mathsf{Paillier\text{-}enc}_N([k_1 \cdot s \bmod q] + \rho \cdot q)$, where $s$ is the value from the signature received from $\mathcal{F}_{\mathsf{det\text{-}ecdsa}}$,
8. $\mathcal{S}$ generates a simulated proof $\pi_c$ that the message $c$ is generated correctly.
9. $\mathcal{S}$ internally hands $(c, \pi_c)$ to $\mathcal{A}$.

The only difference between the view of $\mathcal{A}$ in a real execution and in the simulation in the $\mathcal{F}_{\mathsf{eVRF}}$-hybrid model is the way that $c$ and $\pi_c$ are generated. Specifically, $R_2$ is distributed identically in both cases due to the fact that $R$ is randomly generated by $\mathcal{F}_{\mathsf{det\text{-}ecdsa}}$ in the signature generation (once for each $m$) and thus $k_1^{-1} \cdot R$ has the same distribution as $k_2 \cdot G$.

Regarding the ciphertext $c$, in the simulation it is an encryption of the value $[k_1 \cdot s \bmod q] + \rho \cdot q$, whereas in a real execution it is an encryption of the value $s' = k_2^{-1} \cdot (m' + r \cdot (x_1 + x_2)) + \rho \cdot q$, where $\rho \in \mathbb{Z}_{q^2}$ is random (we stress that all additions here are over the *integers* and not mod$q$, except for where it is explicitly stated in the protocol description). The fact that these two distributions of values are statistically close has been shown in the proof of security in [26].

Finally, regarding $\pi_c$, this is indistinguishable by the zero-knowledge property. (Formally, one replaces the generation of $c$ wjth an honest generation, and then the only difference is the proof. This means that the ability to distinguish a real signing execution from a simulated one can be translated into the ability to distinguish a real proof from a simulated one. The other messages in the signing can be executed by the zero-knowledge distinguisher by providing it all secrets as auxiliary input.) This completes the proof of this simulation case.

**Simulating signing – corrupted $P_2$:** The simulator for the signing phase works as follows:

1. Upon input $(\mathsf{Sign}, m, Q, x_2)$, simulator $\mathcal{S}$ sends $(\mathsf{Sign}, m, Q, x_2)$ to $\mathcal{F}_{\mathsf{det\text{-}ecdsa}}$ and receives back a signature $(r, s)$ on message $m$.
2. $\mathcal{S}$ computes the point $R$ from the signature $(r, s)$, using the ECDSA verification procedure.
3. Let $f_2$ be the stored function from the $\mathcal{F}_{\mathsf{eVRF}}$ initialization phase with identifier $\mathsf{sid}$ ($\mathcal{S}$ has this function and $\mathsf{sid}$).
4. $\mathcal{S}$ computes $k_2 = f_2(H(m))$, $R_2 = k_2 \cdot G$, and $R_1 = k_2^{-1} \cdot R$.
5. $\mathcal{S}$ invokes $\mathcal{A}$ with input $(\mathsf{Sign}, m, Q, x_1)$ and simulates functionality $\mathcal{F}_{\mathsf{eVRF}}$ sending $(\mathsf{eval}, \mathsf{sid}, 1, H(m), R_1)$ to $\mathcal{A}$, as the message from $P_1$ to $P_2$.
6. $\mathcal{S}$ receives $(\mathsf{eval}, \mathsf{sid}, 1, H(m))$ from $\mathcal{A}$ as sent to $\mathcal{F}_{\mathsf{eVRF}}$.
7. $\mathcal{S}$ receives $(c, \pi_c)$ from $\mathcal{A}$ as the message to be sent from $P_2$ to $P_1$.

8. $\mathcal{S}$ verifies $\pi_c$, and simulates $P_1$ aborting if it is not accepting (and instructs $\mathcal{F}_{\mathsf{det\text{-}ecdsa}}$ to not provide output to $P_1$).
9. If $\pi_c$ is accepting, then $\mathcal{S}$ instructs $\mathcal{F}_{\mathsf{det\text{-}ecdsa}}$ to provide output to $P_1$ and outputs whatever $\mathcal{A}$ outputs.

In the $\mathcal{F}_{\mathsf{eVRF}}$-hybrid model, the message seen by $P_2$ in the simulation is *identical* to its view in the real execution. Furthermore, there can only be a difference in the result (whether $P_1$ outputs a valid signature $(r, s)$ or aborts) if $\pi_c$ is an accepting proof and yet $c$ was not generated correctly. This contradicts the soundness of the zero-knowledge proof and thus occurs with negligible probability only. This concludes the proof. □

**Security under concurrent composition.** The simulator in the proof is straight-line (no rewinding) assuming straight-line simulation of the zero-knowledge proofs. As such, by [25], the protocol is UC secure assuming "start synchronization" (meaning that all parties have their input before the protocol begins).

**Extensions.** Probablistic signing can be achieved in the same way as for Schnorr, by providing an additional nonce as input. We remark that if the same nonce is used, then the same signature is obtained and there is no negative security ramification. As such, a timestamp or the like can be used, and this should be sufficient. Regarding precomputation of the first message as discussed for Schnorr, for ECDSA this is more problematic since ECDSA itself has no proof in any standard model. As such, it is not possible to justify (in a standard model) that ECDSA remains secure when $m$ can be chosen after $r$ is known.

**Two-round multiparty ECDSA.** We leave the question of achieving two-round *multiparty* ECDSA open. Our techniques can be used to remove a round of communication of commit-and-open in generating $R$. However, existing protocols require more than two rounds irrespective of this step (the protocol of [15] has only three rounds, but their first round requires additional steps beyond just committing to $R_i$ values).

### 4.6 Verifiable and MPC-Friendly Hierarchical Key Derivation

BIP032/BIP044 [38,33] hierarchical-deterministic (HD) key derivation works by deriving multiple keys from a single root secret, utilizing a tree structure. The method includes hardened derivation and normal derivation. A hardened derivation takes a node's private key and path information and applies a pseudorandom function in order to derive a pseduorandom private key for the child node. In contrast, a normal derivation is applied to a node's public key and public path information only. Normal derivations enable anyone to generate new addresses that can be used, given a public key/address. In addition, it is possible to link different keys that have been normally derived from a single key, and delegation on normally derived keys is not possible (since given the private key of one normally derived key, it is possible to find the private key of all of its siblings in the tree). In contrast, hardened derivations can only be computed by the private key owner, different keys derived via hard derivation from a single node cannot be linked, and given the private key of a hardened derived node it is not possible to find the private key of any of its siblings in the tree.

Although BIP032 prescribes a unified method for hardened and normal derivations, *any pseudorandom function* can be used in its place, and this does not affect the public method used for normal derivation. In this section, we propose a new paradigm for hardened derivations using an eVRF

instead of a standard pseudorandom function. Concretely, hardened derivation in BIP032/BIP044 works by applying SHA512 to a node's private key and path information, and the output is used as the child's private key. (The exact details of how this is carried out is not important here, but this is the basic idea.) Instead of using this method, we propose adding an eVRF private key to the node of the tree, and deriving descendants in the tree by applying the eVRF to the path and taking the result as the private key. Concretely, for a given $\mathsf{path}$ (say, determined as in BIP044), we define the key associated with the node for that path by computing $\mathsf{Eval}(k, \mathsf{path}) = (x, Q, \pi)$, and take $x$ to be the private key and $Q$ to be the public key.[11] This guarantees that all hardened keys are pseudorandom, and cannot be linked. In addition, given any hardened derived key, it is not possible to find any other hardened derived key (by uniqueness of the eVRF output), and so hardened-derived keys can be delegated. This therefore makes it a suitable replacement to BIP032 derivation. We will now explain why this is advantageous, and what feature of BIP032 is lost. (We stress that normal derivations remain unchanged. As such, this method is indistinguishable from standard BIP032, since hardened-derived keys are indistinguishable from random in both methods.)

Before proceeding we remark that the use of HD wallets via BIP032/BIP044 is very popular since it enables parties to backup one seed, and to derive many keys from that seed for different purposes.

**Derivation verifiability.** Standard BIP032 derivation does not provide any validation that a public key in the tree has indeed been correctly derived from the initial seed. In contrast, by the properties of an eVRF, the public key of any derived key can be provably validated (since the eVRF outputs the public key and a proof, by definition). This can be useful in many settings. Consider an institutional wallet, for example, where keys are derived and public keys provided externally for deposit. The party transferring the funds to those addresses actually has no way of knowing that they are correct, barring being "told so". This opens the door to phishing and other attacks, where parties are fooled into transferring funds to malicious addresses. However, using our eVRF method, it suffices to generate a certificate on $\mathsf{vk}$ once and for all, and then any address derived can be linked cryptographically to the issuing institution.

**MPC-friendly derivation.** Consider a Blockchain wallet that uses BIP032/ BIP044, while backing up only the seed (almost all such wallets work this way). If one wishes to construct an MPC wallet so that the user's key is split between the user and an institution, then standard BIP032 derivation becomes very expensive. In particular, securely computing SHA512 operations using techniques like garbled circuits is possible, but expensive (especially, over a low bandwidth communication channel). In such cases, an eVRF based hardened derivation can be much more useful. Specifically, each of the user device and server can generate an eVRF instance, backing them both up, and then new keys can be derived by independently computing eVRF output; each party holds its own share of the private key, and the public key is obtained by adding the public output in both eVRF computations. This preserves the property that only the root eVRF keys need to be backed up (since given them it is possible to derive all keys), and each party can work independently and efficiently to derive a key that is additively shared between them. (Multiplicative sharing is possible in the same way.)

Of course, the above method could be achieved by just applying any local pseudorandom function to the path, and having the parties announce the public result to the other ($P_1$ generates $x_1 \leftarrow PRF_{k_1}(\mathsf{path})$ and announces $Q_1 \leftarrow x_1 \cdot G$, and likewise $P_2$). However, a malicious $P_1$ could

[11] We stress that this is different to BIP032 where the input to SHA512 contains private data.

just generate a random $x_1$ that is independent of $k_1$, and this cannot be detected. Such behavior would make the backup of $k_1$ useless, since the private key in this "derivation" cannot be obtained from it. In order to prevent such behavior, we want $P_1$ and $P_2$ to each provide a zero-knowledge proof that $Q_1$ and $Q_2$ are indeed generated correctly from the backed-up root keys. This ensures that the address $Q = Q_1 + Q_2$ can be used safely, since the private keys needed to derive them have been backed up. An eVRF provides exactly this property.

**Delegation of a sub-tree.** We conclude by noting that our method does *not* support delegating an entire subtree of hardened derivations. This is because the root key is needed to carry out any hardened derivation, and revealing this would reveal all keys. As such, a hardened-derived key can be delegated safely, but the party receiving that private key can only carry out normal derivations. This is not a limitation in the way current wallets work, but this difference from the standard BIP032 is worth noting.

## 5 DDH-Based eVRF

We now turn to constructing efficient eVRFs. In this section we show how to construct an eVRF from a classic secure PRF based on the Decision Diffie-Hellman (DDH) assumption. In Section 6 we construct an eVRF using Pallier encryption.

**The DDH PRF.** This PRF $F_{\text{DDH}} = (\mathsf{KeyGen}, \mathsf{Eval})$ is defined with respect to an ensemble of domains and ranges $\{(\mathcal{X}_\lambda, \mathbb{G}_\lambda)\}_{\lambda=1}^{\infty}$, where $\mathbb{G}_\lambda$ is a group of some prime order $s(\lambda)$ with generator $G_\lambda \in \mathbb{G}_\lambda$. In addition, the PRF uses a function-family ensemble $\{H_\lambda : \mathcal{X}_\lambda \to \mathbb{G}_\lambda\}_{\lambda=1}^{\infty}$ and works as follows

- $\mathsf{KeyGen}(1^\lambda) \to k$: output $k \leftarrow\!\!\$\, \mathbb{Z}_{s(\lambda)}$.
- $\mathsf{Eval}^{H_\lambda}(k, x) \to y$: for $x \in \mathcal{X}_\lambda$ output $y \leftarrow k \cdot H_\lambda(x) \in \mathbb{G}_\lambda$.

Naor, Pinkas, and Reingold [31] showed that $F_{\text{DDH}}$ is a secure PRF whenever DDH holds in the groups $\{\mathbb{G}_\lambda\}_{\lambda=1}^{\infty}$ and $H_\lambda$ is sampled unifomly from $\mathcal{O}_{\mathcal{X}_\lambda, \mathbb{G}_\lambda}$, that is, $H_\lambda$ is modeled as a random oracle. Papadopoulos et al. [34] observe that this PRF can be made into a VRF by publishing $\mathsf{vk} := k \cdot G_\lambda$, and attaching to every evaluation $y \leftarrow \mathsf{Eval}(k, x)$ a non-interactive zero-knowledge proof $\pi$ that $\big(G_\lambda, \mathsf{vk}, H_\lambda(x), y\big)$ is a DDH tuple.

From here on, when there is no confusion, we will drop the index $\lambda$ and simply refer to the group $\mathbb{G}_\lambda$ as $\mathbb{G}$. We use $s$ to denote its order and $G$ its generator.

We construct an eVRF by embedding the output of the DDH PRF "in the exponent" of another group. To do so, we will need an explicit representation of the group $\mathbb{G}$ as an elliptic curve group. Such groups are parameterized by a prime field $\mathbb{F}_q$, where $q = q(\lambda)$, along with two scalars $a, b \in \mathbb{F}_q$. The group is the set of all pairs $(x, y) \in \mathbb{F}_q^2$ such that $y^2 = x^3 + ax + b$, along with the point at infinity. The group operation is defined using the cord-and-tangent rule discussed below [7].

Since we embed the group $\mathbb{G}$ in the exponent of another group, it is convenient to introduce the following notion of a group pair, or more precisely, a group pair ensemble.

**Definition 6.** *We say that* $\big\{(\mathbb{G}_{\text{S}}^{(\lambda)}, \mathbb{G}_{\text{T}}^{(\lambda)})\big\}_{\lambda=1}^{\infty}$ *is a* ***group pair ensemble*** *if for every* $(\mathbb{G}_{\text{S}}, \mathbb{G}_{\text{T}})$ *in the ensemble we have that*

– *The group $\mathbb{G}_{\mathrm{T}}$, called the **target group**, has some prime order $q$. We use $\boldsymbol{G} = (G_{\mathrm{T},1}, \ldots, G_{\mathrm{T,n}})$ to denote a vector of $n$ generators in $\mathbb{G}_{\mathrm{T}}$. For a vector $\boldsymbol{x} \in \mathbb{F}_q^n$ we write*

$$\langle \boldsymbol{x}, \boldsymbol{G} \rangle = x_1 G_{\mathrm{T},1} + \cdots + x_n G_{\mathrm{T,n}} \in \mathbb{G}_{\mathrm{T}}.$$

– *The group $\mathbb{G}_{\mathrm{S}}$, called the **source group**, is a group of some prime order $s$ with generator $G_{\mathrm{S}} \in \mathbb{G}_{\mathrm{S}}$. This group $\mathbb{G}_{\mathrm{S}}$ is a group of points of an elliptic curve defined over the field $\mathbb{F}_q$, where $q$ is the order of $\mathbb{G}_{\mathrm{T}}$. Let $\mathbb{G}_{\mathrm{S}}^* := \mathbb{G}_{\mathrm{S}} \setminus \{0\}$. Elements in $\mathbb{G}_{\mathrm{S}}^*$ are represented as pairs in $\mathbb{F}_q^2$ so that $\mathbb{G}_{\mathrm{S}}^* \subseteq \mathbb{F}_q^2$.*

*We say that the group pair ensemble is **secure** if DLOG holds in $\{\mathbb{G}_{\mathrm{T}}^{(\lambda)}\}_{\lambda=1}^{\infty}$ and DDH holds in $\{\mathbb{G}_{\mathrm{S}}^{(\lambda)}\}_{\lambda=1}^{\infty}$. To simplify the notation, we will often drop the index $\lambda$ and say that $(\mathbb{G}_{\mathrm{S}}, \mathbb{G}_{\mathrm{T}})$ is a **group pair** or a **secure group pair**.*

In our eVRF construction, we will use the source group, $\mathbb{G}_{\mathrm{S}}$, for the output of the DDH PRF. We will use the target group, $\mathbb{G}_{\mathrm{T}}$, to hide the PRF output "in the exponent" of an element in $\mathbb{G}_{\mathrm{T}}$. To do so, we need an explicit representation of elements in $\mathbb{G}_{\mathrm{S}}$. Requiring that $\mathbb{G}_{\mathrm{S}}$ is an elliptic curve group is sufficient.

The challenge is to design a proof system that proves that the DDH PRF was evaluated correctly, despite being given the value of the PRF in the exponent. Towards this goal, we will make use of the following instance-witness relation $\mathcal{R}_H$ defined with respect to a function $H : \mathcal{X} \to \mathbb{G}_{\mathrm{S}}$ as

$$\begin{aligned} \mathcal{R}_H := \Big\{ \big((Q, x, Y)\ ;\ k\big) \Big\} \subseteq (\mathbb{G}_{\mathrm{T}} \times \mathcal{X} \times \mathbb{G}_{\mathrm{T}}) \times [s-1] \quad \text{where} \\ (1)\ \ Q = k \cdot G_{\mathrm{T},1}, \\ (2)\ \ Y = x_P \cdot G_{\mathrm{T},2} \text{ for } P = (x_P, y_P) := k \cdot H(x) \in \mathbb{G}_{\mathrm{S}}^* \subseteq \mathbb{F}_q^2. \end{aligned} \tag{1}$$

Here we are treating $G_{\mathrm{T},1}$ and $G_{\mathrm{T},2}$ as generators of $\mathbb{G}_{\mathrm{T}}$ that are part of the description of the group. As usual, we let $\mathcal{L}_{\mathcal{R}_H}$ denote the language of all triples $(Q, x, Y)$ for which there exists a $k \in [s-1]$ such that $\mathcal{R}_H\big((Q, x, Y); k\big)$ is true.

In addition, to uplift the security of our eVRF from a game-based one (Definition 4) to realizing the ideal functionality (Definition 5), we require a non-interactive zero-knowledge proof-of-knowledge (ZK-POK) for the discrete log relation

$$\mathcal{R}_{\mathrm{dlog}} := \Big\{ \big(Q; k\big)\ :\ Q = k \cdot G_{\mathrm{T},1} \Big\} \subseteq \mathbb{G}_{\mathrm{T}} \times [s-1] \tag{2}$$

One can use Schnorr's protocol [36] for $\mathcal{R}_{\mathrm{dlog}}$. However, to realize the ideal functionality we require a straight line (non-rewinding) knowledge extractor, for example, as presented by Fischlin [18].

**The DDH eVRF.** We can now present our eVRF. The construction uses a non-interactive zero-knowledge argument system $(\mathsf{P}, \mathsf{V})$ for the relation $\mathcal{R}_H$ from (1). We will present the required argument system in Section 5.1 below. It also uses the Schnorr ZK-POK $(\mathsf{P}_{\mathrm{dlog}}, \mathsf{V}_{\mathrm{dlog}})$ for the relation $\mathcal{R}_{\mathrm{dlog}}$ from (2).

**Definition 7 (DDH-based eVRF).** *Let $(\mathbb{G}_{\mathrm{S}}, \mathbb{G}_{\mathrm{T}})$ be a group pair where $s$ is the size of $\mathbb{G}_{\mathrm{S}}$ and $q$ is the size of $\mathbb{G}_{\mathrm{T}}$. Let $G_{\mathrm{T},1}, G_{\mathrm{T},2}$ be two generators of $\mathbb{G}_{\mathrm{T}}$. Let $H$ be a function $H : \mathcal{X} \to \mathbb{G}_{\mathrm{S}}^*$, where $\mathbb{G}_{\mathrm{S}}^* := \mathbb{G}_{\mathrm{S}} \setminus \{0\}$. The **DDH eVRF** with domain $\mathcal{X}$ and range $\mathbb{G}_{\mathrm{T}}$ is defined as follows:*

– $\mathsf{KeyGen}^H(1^\lambda)$*: Sample $k \leftarrow\!\!\$\ [s-1]$ and set $Q \leftarrow k \cdot G_{\mathrm{T},1}$. Use the prover $\mathsf{P}_{dlog}$ for $\mathcal{R}_{dlog}$ to generate a proof $\pi_Q \leftarrow\!\!\$\ \mathsf{P}_{dlog}(Q, k)$ and set $\mathsf{vk} := (Q, \pi_Q)$. Output $(k, \mathsf{vk})$.*

– $\mathsf{Eval}^H(k,x)$: *for* $x \in \mathcal{X}$, *let* $P \leftarrow k \cdot H(x) \in \mathbb{G}_{\mathrm{S}}^*$.

$$\text{with } P = (x_P, y_P) \subseteq \mathbb{F}_q^2 \quad \text{set} \quad y \leftarrow x_P \in \mathbb{F}_q \quad \text{and} \quad Y \leftarrow y \cdot G_{\mathrm{T},2} \in \mathbb{G}_{\mathrm{T}}.$$

*Next, run the prover* $\mathsf{P}$ *for* $\mathcal{R}_H$ *to construct a proof* $\pi$ *that the triple* $\big(\mathsf{vk}, x, Y\big)$ *is in the language* $\mathcal{L}_{\mathcal{R}_H}$ *from* (1). *The proof system for* $\mathcal{R}_H$ *is described in Section 5.1 below. Output* $(Y, \pi, y)$.

– $\mathsf{Verify}^H(\mathsf{vk}, x, Y, \pi)$: *for* $\mathsf{vk} = (Q, \pi_Q)$, *the algorithm accepts if (1)* $\pi$ *is a valid proof that* $(Q, x, Y)$ *is in* $\mathcal{L}_{\mathcal{R}_H}$, *and (2)* $\pi_Q$ *is a valid proof for* $Q$ *as an instance of* $\mathcal{R}_{dlog}$.

At the heart of this eVRF construction is the non-interactive zero-knowledge proof for the language $\mathcal{L}_{\mathcal{R}_H}$. However, before we develop this proof system, let us first briefly argue that the eVRF is secure when $H : \mathcal{X} \to \mathbb{G}_{\mathrm{S}}^*$ is modeled as a random oracle.

**Theorem 13 (eVRF security).** *Let* $(\mathbb{G}_{\mathrm{S}}, \mathbb{G}_{\mathrm{T}})$ *be a secure group pair. Let* $(\mathsf{P}, \mathsf{V})$ *be a non-interactive zero-knowledge argument system for the relation* $\mathcal{R}_H$ *from* (1). *Then the eVRF in Definition 7 is a secure eVRF (as in Definition 4) with respect to the domain/range* $(\mathcal{X}, \mathbb{G}_{\mathrm{T}})$ *and function family* $\mathcal{O}_{\mathcal{X},\mathbb{G}_{\mathrm{S}}^*}$.

*Proof. Consistency* holds by construction. *Pseudorandomness* follows from the fact that the DDH PRF is a secure PRF when DDH holds in $\mathbb{G}_{\mathrm{S}}$ and $H$ is sampled at random from $\mathcal{O}_{\mathcal{X},\mathbb{G}_{\mathrm{S}}^*}$. *Verifiability as a VRF* follows from the soundness of the proof system for $\mathcal{R}_H$. *Simulatability as a VRF* follows from the zero knowledge property of the proof system for $\mathcal{R}_H$. □

Looking ahead, soundness of our proof system for $\mathcal{R}_H$ relies on the hardness of discrete log in $\mathbb{G}_{\mathrm{T}}$ and the knowledge soundness of the proof system for $\mathcal{R}_{\mathrm{dlog}}$. It also relies on $G_{\mathrm{T},1}, G_{\mathrm{T},2}$ being random generators of $\mathbb{G}_{\mathrm{T}}$ so that there is no known discrete log relation between them.

### 5.1 An Argument System for the Relation $\mathcal{R}_H$

To complete the construction, we need an efficient non-interactive zero-knowledge argument for the relation $\mathcal{R}_H$ from (1). There are several ways to proceed. One option is to use a generic zkSNARK [4] to produce a succinct proof. However, since $\mathcal{R}_H$ uses arithmetic in both $\mathbb{G}_{\mathrm{S}}$ and $\mathbb{G}_{\mathrm{T}}$, this will require non-native arithmetic in the zkSNARK, which will result in an efficient prover.

Another option is to use the Bulletproofs argument system [8,10], which is especially well suited for proving statements about $\mathbb{F}_q$ field elements that are given "in the exponent." This is precisely what is needed for the relation $\mathcal{R}_H$: the verifier is given $k$ and $x'$ in the exponent — they are provided as $Q = k \cdot G_{\mathrm{T},1}$ and $Y = x_P \cdot G_{\mathrm{T},2}$ — along with $x \in \mathcal{X}$, and we need a proof that $x_P$ is the $x$-coordinate of $P := k \cdot H(x) \in \mathbb{G}_{\mathrm{S}}$. We show how to use Bulletproofs to construct an efficient argument system for $\mathcal{R}_H$. The resulting the proof size is only $O(\log \log s)$ group elements, and prover and verifier times are dominated by a $O(\log s)$ multi-scalar multiplication in $\mathbb{G}_{\mathrm{T}}$.

**A brief overview of Bulletproofs.** Bünz [9, §2.6] shows that Bulletproofs give an argument system for a rank-1 constraint system (R1CS), when the statement is given in the exponent. Specifically, Bulletproofs is well suited as an argument system for the following exponent R1CS relation:

$$\begin{aligned} \mathcal{R}_{\mathrm{eR1CS}} := \Big\{ \big(A, B, C, T\big) \;;\; \big(\boldsymbol{x}, \boldsymbol{w}\big) \Big\} \quad & \text{where} \\ (1)\quad & A, B, C \in \mathbb{F}_q^{n \times m}, \quad T \in \mathbb{G}_{\mathrm{T}}, \quad \boldsymbol{x} \in \mathbb{F}_q^r, \quad \boldsymbol{w} \in \mathbb{F}_q^{m-r}, \\ (2)\quad & (A\boldsymbol{z}) \circ (B\boldsymbol{z}) = (C\boldsymbol{z}) \quad \text{where} \quad \boldsymbol{z} := (\boldsymbol{x}, \boldsymbol{w}) \in \mathbb{F}_q^m, \\ (3)\quad & T = \langle \boldsymbol{x}, \boldsymbol{G} \rangle. \end{aligned} \tag{3}$$

Here $\boldsymbol{G} \in \mathbb{G}_{\mathrm{T}}^r$ is a public tuple of $r$ generators of $\mathbb{G}_{\mathrm{T}}$. The notation $\boldsymbol{u} \circ \boldsymbol{v}$ used on line (2) refers to the component-wise multiplication of the vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ (also called the Hadamard product of $\boldsymbol{u}$ and $\boldsymbol{v}$).

Note that the R1CS statement $\boldsymbol{x} \in \mathbb{F}_q^r$ is provided as a Pedersen commitment $T = \langle \boldsymbol{x}, \boldsymbol{G} \rangle$, exactly as in our settings. Indeed, for an $\mathcal{R}_H$-instance $(Q, x, Y)$ we will set $T := Q + Y \in \mathbb{G}_{\mathrm{T}}$. The vector $\boldsymbol{z}$ on line (2) is often called the **extended witness**. Each row in the matrices $A, B, C$ is called a **constraint**, so that the R1CS above has $n$ constraints. In the Bulletproofs argument, the prover sends to the verifier a Pedersen commitment $T' := \langle \hat{\boldsymbol{z}}, \boldsymbol{G}_z \rangle$ for a vector $\hat{\boldsymbol{z}}$ derived from $\boldsymbol{z}$. An inner-product argument is then used to prove that $\boldsymbol{z} := (\boldsymbol{x}, \boldsymbol{w})$ satisfies the condition on line (2).

The Bulletproofs argument system is complete, computationally sound, and zero knowledge. Here computational soundness means that either the system is sound, or there is a PPT algorithm that can find a non-trivial linear relation among the generators in $(\boldsymbol{G}, \boldsymbol{G}_z)$. The latter implies that discrete log is easy in $\mathbb{G}_{\mathrm{T}}$. The argument system can be made non-interactive using the Fiat-Shamir transform, and retains its soundness and zero knowledge properties in the random oracle model [1].

The length of the proof is $2\log_2(n+m) + 4$ group elements in $\mathbb{G}_{\mathrm{T}}$. The running times of the verifier is dominated by the time to compute a multi-scalar multiplication (MSM) for a vector dimension about $2(n+m)$. Using Pippenger's algorithm [35], computing such an MSM is faster than computing the exponentiations one by one. In addition, the Bulletproofs verifier can batch verify multiple proofs at once much faster than verifying the proofs one at a time [10]. We summarize these facts in the following theorem.

**Theorem 14 ([9]).** *Bulletproofs is a zero knowledge non-interactive argument system for the relation $\mathcal{R}_{eR1CS}$ in the random oracle model, assuming discrete log in $\mathbb{G}_{\mathrm{T}}$ is hard. The length of the proof is $2\lceil \log_2(n+m) \rceil + 3$ group elements in $\mathbb{G}_{\mathrm{T}}$.*

**The proof system for $\mathcal{R}_H$.** Given the above, it remains to design an efficient rank-1 constraint system (R1CS) — namely three matrices $A, B, C \in \mathbb{F}_q^{n \times m}$ — for the relation $\mathcal{R}_H$ from (1). Consider an $\mathcal{R}_H$ instance $\big((Q, x, Y)\ ;\ k\big)$ where $k$ is in $[s-1]$. Let $(k_0, k_1, \ldots, k_\ell) \in \{0,1\}^{\ell+1}$ be the binary representation of $k$, so that $k = \sum_{i=0}^{\ell} 2^i \cdot k_i$. In addition, we use a fixed sequence of elements $c_0, \ldots, c_\ell \in \mathbb{Z}_s$ satisfying

$$\forall i \in [\ell - 1]: \quad \sum_{j=0}^{i-1} c_j \notin \{0, \pm c_i\} \qquad \text{and} \qquad \sum_{j=0}^{\ell} c_j = 0. \tag{4}$$

For example, when $s > \ell^2$ one can set $c_i := (i+2)$ for $i = 0, \ldots, \ell - 1$ and $c_\ell := -\binom{\ell+2}{2} + 1$.

Let $X := H(x)$. Our plan for proving that $(Q, x, Y)$ is in $\mathcal{L}_{\mathcal{R}_H}$ is to have the prover compute a sequence of elements $P_0, \ldots, P_\ell \in \mathbb{G}_{\mathrm{S}}$ defined by

$$\begin{cases} P_0 := k_0 \cdot X + c_0 \cdot G_{\mathrm{S}} \\ P_i := P_{i-1} + \Delta_i \quad \text{where} \quad \Delta_i := (2^i k_i) \cdot X + c_i \cdot G_{\mathrm{S}} \quad (\text{for } i = 1, \ldots, \ell). \end{cases} \tag{5}$$

These points will become part of the extended witness $\boldsymbol{z}$ for our R1CS program. Observe that because $\sum_{i=0}^{\ell} 2^i k_i = k$ and $\sum_{j=0}^{\ell} c_j = 0$, the final point $P_\ell$ satisfies $P_\ell = k \cdot X \in \mathbb{G}_{\mathrm{S}}$. Now the R1CS program only needs to check that if $Y = x \cdot G_{\mathrm{T},2}$, then the $x$-coordinate of $P_\ell$ is equal to $x$, which is just one constraint in R1CS. The main challenge is to verify that the point $P_\ell$ was constructed

correctly. We will do so by verifying inductively that $P_i$ is correct given that all $P_0, \ldots, P_{i-1}$ are correct (for all $i = 1, \ldots, \ell$).

Looking ahead, the purpose of the term $c_i G_S$ in the definition of $\Delta_i$ in (5) is to ensure that $\Delta_i$ is not equal to $\pm P_{i-1}$. This ensures that none of the $P_i$ are the point at infinity in $\mathbb{G}_S$, and that the addition always adds distinct points in $\mathbb{G}_S$. This lets us always use the "cord" rule for addition, and never need the "tangent" rule.

Let us describe the R1CS program for $\mathcal{L}_{\mathcal{R}_H}$ in more detail. We begin by describing the extended witness $\boldsymbol{z}$. Recall that $(k_0, k_1, \ldots, k_\ell)$ is the binary representation of the secret key $k \in [s-1]$. For $i = 0, \ldots, \ell$ we let $P_i = (x_{P_i}, y_{P_i}) \in \mathbb{G}_S^* \subseteq \mathbb{F}_q^2$ be the points constructed in (5). Further, define

$$\boldsymbol{w}_i := \Big(k_i,\ x_{P_i}, x_{P_i}^2, x_{P_i}^3,\ y_{P_i}, y_{P_i}^2,\ t_1, t_2\Big)^\intercal \in \mathbb{F}_q^8 \quad \text{for } i = 1, \ldots, \ell. \tag{6}$$

We will explain what are $t_1, t_2 \in \mathbb{F}_q$ soon below. Then, with $P = (x_P, y_P) = kX \in \mathbb{G}_S$, the extended witness is defined as the column vector

$$\boldsymbol{z} := \Big(\underbrace{1, k, x_P}_{\substack{\text{the R1CS}\\ \text{statement}}},\quad k_0, x_{P_0}, y_{P_0},\quad \boldsymbol{w}_1, \ldots, \boldsymbol{w}_\ell\Big)^\intercal \in \mathbb{F}_q^{8\ell+6}. \tag{7}$$

Now, the R1CS program, which consists of three matrices $A, B, C \in \mathbb{F}_q^{n \times m}$, needs to verify three claims:

- *claim 1:* $k_0, \ldots, k_\ell$ is the binary representation of $k$, that is $k = \sum_{i=0}^{\ell} 2^i k_i$ and $k_i \in \{0, 1\}$ for $i = 0, \ldots, \ell$.
- *claim 2:* The points $P_0, \ldots, P_\ell$ are constructed according to (5).
- *claim 3:* The point $P_\ell = (x_{P_\ell}, y_{P_\ell})$ satisfies $x_{P_\ell} = x_P$, where $P := kX$.

If all three claims hold then the verifier is convinced that $(Q, x, Y)$ is in $\mathcal{L}_{\mathcal{R}_H}$.

Claims 1 and 3 are easy to check in an R1CS: checking that $k = \sum_{i=0}^{\ell} 2^i k_i$ takes one constraint (i.e., one row in $A, B, C$); checking that $k_i \in \{0, 1\}$ for $i = 0, \ldots, \ell$ is done by checking that $k_i(1 - k_i) = 0$, and this takes $\ell + 1$ constraints, one for each $k_i$; checking that $x_{P_\ell} = x_P$ takes one constraint. Hence, Claims 1 and 3 require a total of $\ell + 3$ constraints in $A, B, C$.

Checking Claim 2 in R1CS is more interesting. First, note that while the verifier has $X, G_S \in \mathbb{G}_S$, it does not know the bits $k_i$ of $k$ and therefore cannot construct the points $\Delta_i$ in (5) by itself. However, we observe that given $X$ and $G_S$, all the $\Delta_i \in \mathbb{G}_S$ can be expressed as a public linear function of $k_i$. Indeed, since $k_i$ is binary, we know that $\Delta_i$ takes one of two values:

$$\Delta_i = \Delta := 2^i X + c_i G_S \qquad \text{or} \qquad \Delta_i = \Delta' := c_i G_S.$$

Let $(x, y) \in \mathbb{F}_q^2$ be the elliptic curve point representing $\Delta$ and let $(x', y') \in \mathbb{F}_q^2$ be the point representing $\Delta'$. Then we can express $\Delta_i$ as

$$\Delta_i = k_i(x - x', y - y') + (x', y') = (k_i \delta_x + x',\quad k_i \delta_y + y') \in \mathbb{F}_q^2 \tag{8}$$

where $\delta_x := x - x'$ and $\delta_y := y - y'$. The verifier can construct $x', y', \delta_x, \delta_y \in \mathbb{F}_q$ on its own. Hence, $\Delta_i$ can be expressed as a public linear function of $k_i$. Since $P_o = \Delta_0$ this method also lets us express $P_0$ as a public linear function of $k_0$.

Now, to verify claim 2, the R1CS program will verify that $P_i = P_{i-1} + \Delta_i$ for all $i = 1, \ldots, \ell$. The program does so by checking two things:

– First, for $i = 1, \ldots, \ell$ verify that the point $P_i = (x_{P_i}, y_{P_i})$ in $\boldsymbol{z}$ is a point in $\mathbb{G}_{\mathrm{S}}^*$. That is, $(x_{P_i}, y_{P_i}) \in \mathbb{F}_q^2$ satisfies $y_{P_i}^2 = x_{P_i}^3 + a x_{P_i} + b$ where $a$ and $b$ are the constants that define the curve $\mathbb{G}_{\mathrm{S}}$. This takes four constraints in the matrices $A, B, C$: two constraints to verify that $x_{P_i}^3$ in $\boldsymbol{z}$ is computed correctly (i.e., it is the cube of $x_{P_i}$); one constraint to verify that $y_{P_i}^2$ in $\boldsymbol{z}$ is computed correctly (i.e., it is the square of $y_{P_i}$); and one linear constraint to verify the elliptic curve relation $y_{P_i}^2 = x_{P_i}^3 + a x_{P_i} + b$.
– Second, for $i = 1, \ldots, \ell$ verify that the points $P_{i-1}$, $\Delta_i$ and $-P_i = (x_{P_i}, -y_{P_i})$ are co-linear. When $P_{i-1} \neq \pm\Delta_i$ this is sufficient to prove that $P_i = P_{i-1} + \Delta_i$ in $\mathbb{G}_{\mathrm{S}}$. Moreover, we show in Theorem 15 that indeed $P_{i-1} \neq \pm\Delta_i$ must always hold. To check that $P_{i-1}$, $\Delta_i$ and $-P_i$ are co-linear, the R1CS program verifies that the slope of the line through $P_{i-1}$ and $-P_i$ is equal to the slope of the line through $\Delta_i$ and $-P_i$. That is, the program verifies that

$$(y_{P_{i-1}} + y_{P_i}) \cdot (x_{\Delta_i} - x_{P_i}) = (y_{\Delta_i} + y_{P_i}) \cdot (x_{P_{i-1}} - x_{P_i}) \tag{9}$$

  This is where the values $t_1$ and $t_2$ in (6) are used. The R1CS verifies that $t_1$ is equal to the left hand side of (9), that $t_2$ is equal to the right hand side, and that $t_1 = t_2$. This takes a total of three constraints in $A, B, C$. Recall that $x_{\Delta_i}$ and $y_{\Delta_i}$ used in (9) are expressed as a linear function of $k_i$ using (8).

The explicit matrices $A, B, C$ are shown in Figure 1 in the appendix. Together, the two checks above prove that $P_1, \ldots, P_\ell$ in $\boldsymbol{z}$ are computed as in (5). It remains to verify that $P_0$ is constructed correctly, and this is done using (8), which takes two constraints.

**Putting it all together.** Let $(Q, x, Y)$ be a $\mathcal{R}_H$ instance. Let $(A, B, C)$ be the R1CS constructed from $H(x)$ and $G_{\mathrm{S}}$ as described above. Then the proof system for $\mathcal{R}_H$ works as follows: the prover sets $T := Q + Y$ and runs the Bulletproofs prover for the $\mathcal{R}_{\mathrm{eR1CS}}$ instance $(A, B, C, T)$. It outputs the resulting proof $\pi$. The verifier also sets $T := Q + Y$ and runs the Bulletproofs $\mathcal{R}_{\mathrm{eR1CS}}$ verifier for the instance $(A, B, C, T)$ with proof $\pi$. This is the entire proof system for $\mathcal{R}_H$.

The performance of the system is determined by the dimensions of the matrices $A, B, C$. All the linear constraints in the R1CS $(A, B, C)$ can be collapsed into a single constraint by taking a random linear combination of the constraints using verifier randomness. Consequently, the total number of the constraints in our R1CS is: $(\ell+1)$ constraints for claims 1 and 3; another $3\ell$ constraints to verify that all $P_i$ are in $\mathbb{G}_{\mathrm{S}}^*$; and $2\ell$ constraints to verify co-linearity. This is a total for $6\ell + 1$ constraints, plus one more constraint to verify all the linear constraints at once. An observation in Figure 1 in the appendix reduces the number of constraints to $5\ell + 2$. Therefore, the matrices $A, B, C \in \mathbb{F}_q^{n \times m}$ have dimension $n = 5\ell + 2$ and $m = 8\ell + 6$.

The proof system for $\mathcal{R}_H$ is complete and inherits its zero knowledge property from the proof for $\mathcal{L}_{\mathcal{R}_{\mathrm{eR1CS}}}$. It remains to prove computational soundness, which is proved in the following theorem.

**Theorem 15.** *For a hash function $H : \mathcal{X} \to \mathbb{G}_{\mathrm{S}}$, let $(Q, x, Y)$ be an instance for $\mathcal{L}_{\mathcal{R}_H}$. Set $T := Q + Y$ and let $(A, B, C)$ be the R1CS constructed in this section from $H(x)$ and $G_{\mathrm{S}}$ in $\mathbb{G}_{\mathrm{S}}$. Then*

$$(A, B, C, T) \in \mathcal{L}_{\mathcal{R}_{eR1CS}} \qquad \Longrightarrow \qquad (Q, x, Y) \in \mathcal{L}_{\mathcal{R}_H},$$

*assuming discrete log is hard in $\mathbb{G}_{\mathrm{S}}$, and $H$ is modeled as a random oracle.*

We will prove Theorem 15 in a minute. First, by combining Theorems 14 and 15 we obtain the following corollary.

**Corollary 3.** *Bulletproofs gives a zero-knowledge non-interactive argument system for $\mathcal{R}_H$ in the random oracle model, assuming discrete log is hard in both $\mathbb{G}_{\mathrm{S}}$ and $\mathbb{G}_{\mathrm{T}}$. The length of the proof is $2\lceil \log_2(\ell) \rceil + 12$ group elements in $\mathbb{G}_{\mathrm{T}}$.*

Concretely, for $\ell = 256$ we obtain a non-interactive proof that contains about 28 group elements in $\mathbb{G}_{\mathrm{T}}$. For a 256-bit elliptic curve this comes out to about 900 bytes.

**Proof of Theorem 15.** We already explained that our R1CS $(A, B, C)$ verifies that the three claims about the points $P_0, \ldots, P_\ell$ hold, and this implies that $(Q, x, Y)$ is in $\mathcal{L}_{\mathcal{R}_H}$. The only part that remains to argue is that when verifying that $P_i = P_{i-1} + \Delta_i$ it suffices to use the cord rule, as we did in (9). That is, we need to argue that $P_{i-1} \neq \pm\Delta_i$ for $i = 1, \ldots, \ell$. To do so, consider an adversary $\mathcal{A}$ that outputs $(\mathsf{vk}, x, Y, \pi)$ such that (1) the DDH eVRF accepts, namely $\mathsf{Verify}(\mathsf{vk}, x, Y, \pi) = \mathsf{true}$, and (2) if $\mathsf{vk} = (Q, \pi_Q)$ and $Q = kG_{\mathrm{T},1}$, then the derived points $P_0, \ldots, P_\ell$ satisfy that $P_{i-1} = \pm\Delta_i$ for some $i$. We use this adversary $\mathcal{A}$ to break discrete log in $\mathbb{G}_{\mathrm{S}}$.

Let us construct an algorithm $\mathcal{B}$ that computes discrete log in $\mathbb{G}_{\mathrm{S}}$. This $\mathcal{B}$ takes as input $R \in \mathbb{G}_{\mathrm{S}}$ and needs to output an $\alpha \in \mathbb{Z}_s$ such that $R = \alpha G_{\mathrm{S}}$. Algorithm $\mathcal{B}$ runs $\mathcal{A}$ and responds to its random oracle queries for $H(x_i)$ by choosing a random $r_i \leftarrow\!\!\$\, \mathbb{Z}_s$ and responding with $H(x_i) = r_i R$.

Eventually, algorithm $\mathcal{A}$ outputs $(\mathsf{vk}, x, Y, \pi)$, where $\mathsf{vk} = (Q, \pi_Q)$. Since the eVRF $\mathsf{Verify}(\mathsf{vk}, x, Y, \pi)$ accepts we obtain three things:

- First, $\mathcal{B}$ can run the extractor for the proof system for $\mathcal{R}_{\mathrm{dlog}}$ to extract from $\mathcal{A}$ an integer $k \in [s-1]$ such that $Q = kX$. As usual, we let $(k_0, \ldots, k_\ell)$ be the binary representation of $k$.
- Second, if we set $T := Q + Y$, then we know that $(A, B, C, T)$ is in $\mathcal{L}_{\mathcal{R}_{\mathrm{eR1CS}}}$. This is because the Bulletproofs proof $\pi$ is accepted by the verifier.
- Third, since $\mathcal{A}$ must have queried for $H(x)$, it follows that $\mathcal{B}$ knows an $r \in \mathbb{Z}_s$ such that $H(x) = rR$.

Next, by assumption on $\mathcal{A}$ we know that the points $P_0, \ldots, P_\ell$, derived from $H(x)$ and $G_{\mathrm{S}}$ via (5) satisfy that $P_0 = 0$ or $P_{i-1} = \pm\Delta_i$ for some $i \in [\ell]$. If $P_0 = 0$ then $k_0(rR) + c_0 G_{\mathrm{S}} = 0$. Then since $c_0 \neq 0$ it must be that $k_0 r \neq 0$, and this immediately reveals the discrete log $\alpha$ such that $R = \alpha G_{\mathrm{S}}$.

Next, if $P_0 \neq 0$, let $i^*$ be the smallest index $i$ in $[\ell]$ such that $P_{i^*-1} = \pm\Delta_{i^*}$. For now, let us assume that $P_{i^*-1} = -\Delta_{i^*}$. The case $P_{i^*-1} = \Delta_{i^*}$ is handled the same way. Since $(A, B, C, T)$ is in $\mathcal{L}_{\mathcal{R}_{\mathrm{eR1CS}}}$ we know that $P_0, \ldots, P_{i^*-1}$ must be computed as in (5). Therefore,

$$2^{i^*} k_{i^*}(rR) + c_{i^*} G_{\mathrm{S}} = \Delta_{i^*} = -P_{i^*-1} = -\left[ \sum_{j=0}^{i^*-1} (2^j k_j)(rR) + \sum_{j=0}^{i^*-1} c_j G_{\mathrm{S}} \right]$$

which leads to

$$r \sum_{j=0}^{i^*} (2^j k_j) \cdot R = \sum_{j=0}^{i^*} c_j \cdot G_{\mathrm{S}}. \tag{10}$$

When $i^* < \ell$ we know by construction of $c_i$ in (4) that $\sum_{j=0}^{i^*} c_j \neq 0$. It follows that the left hand side is non-zero, and then (10) reveals the discrete log $\alpha$. The case $i^* = \ell$ is not possible because then the left hand side of (10) is non-zero since $k \neq 0$, but the right hand side is zero because $c_0 + \cdots + c_\ell = 0$. The case $P_{i^*-1} = \Delta_{i^*}$ is the same and relies on the fact that $c_{i^*} \neq \sum_{j=0}^{i^*-1} c_j$.

Hence, we conclude that if discrete log in $\mathbb{G}_{\mathrm{S}}$ is hard, then $P_0 \neq 0$ and $P_{i-1} \neq \pm\Delta_i$ for all $i \in [\ell]$, as required for (9) to properly verify addition in $\mathbb{G}_{\mathrm{S}}$. This completes the proof of the theorem. □

*Remark 1 (an optimization).* It is not difficult to generalize (8) and process two bits of the key $k$ at every iteration, instead of one bit as in (8). This will halve the number of iterations, at the cost of two additional constraints per iteration.

**Practical considerations.** For the applications to ECDSA discuss in Section 4, the target group $\mathbb{G}_{\mathrm{T}}$ needs to be the standard group of points on the elliptic curve Secp256k1. This curve is defined by the elliptic curve equation $y^2 = x^3 + 7$ in $\mathbb{F}_p$ for a specific prime $p$. This curve has $q$ point in $\mathbb{F}_p$ for some prime $q$. Remarkably, for such curves, a theorem due to Silverman and Stange [37, Cor. 22], shows that the same curve $y^2 = x^3 + 7$, but this time defined over $\mathbb{F}_q$, has prime order $p$. Hence, we can take as our source group the curve $y^2 = x^3 + 7$ defined over $\mathbb{F}_q$.

Suppose that instead we use the curve Ed25519 [3] as the target group $\mathbb{G}_{\mathrm{T}}$, as needed for EdDSA. The curve is defined over $\mathbb{F}_p$ where $p := 2^{255} - 19$ and has order $8q$ for some prime $q$. In this case we would choose some prime order elliptic curve defined over $\mathbb{F}_q$ and use it as our source group.

The running time of the verifier is dominated by the time to do a multiscalar multiplication (MSM) of a vector of dimension $2 \times (13\ell)$. For $\ell = 256$ this gives an MSM of length about 6,600. This drops to about 4,000 using Remark 1. The running time of the prover is comparable.

The cost of multiscalar multiplications (MSM) for these dimensions is about a sixth of a sequence multiplications done one at a time. In addition, the bases are all fixed ahead of time, and so the multiplications can be sped up pre-computing the required tables. We therefore estimate the cost of the MSM by considering the cost of multiplications and dividing by six. A crypto library for the curve secp256k1 computes 140,000 generator multiplications per second, and for Ed25519 it computes 55,000 generator multiplications per second (running a single thread on a 2.3 GHz 8-Core Intel Core i9). This yields an estimated running time for proving and verifying of approximately just 5ms for secp256k1 and 14ms for Ed25519. Of course, this can be further accelerated by employing multiple threads in parallel.

## 6 Paillier-Based eVRF

In this section, we use the Paillier encryption scheme [32] to present a simple construction of eVRF. Recall that the Paillier encryption scheme is additively homomorphic over the integers, and is CPA assuming the *decisional composite residuosity assumption (DCRA)* holds.

Let us us start with a high-level presentation of the construction. Fix a "target group" $\mathbb{G}$ for the eVRF and a generator $G$ of $\mathbb{G}$. As a warm-up, assume we could have efficiently computed discrete logarithm in $\mathbb{G}$: for $Y \in \mathbb{G}$, efficiently compute $y \leftarrow \log Y$ with $y \cdot G = Y$. Let $H$ be a random oracle mapping arbitrary strings to uniform values in $\mathbb{G}$. Then $\mathsf{Eval}(x) := (\log H(x), H(x), \perp)$ would be a perfect eVRF. (The verification procedure would simply assert that $H(x) = Y$.)

Since, fortunately, we cannot efficiently compute the discrete log in groups of interest, we take a similar approach using an intermediate "group" in which we can compute "discrete log", and then map the result into $\mathbb{G}$. Specifically, we use Paillier ciphertexts as the intermediate "group". The key generation algorithm samples a Paillier public and secret key pair $(\mathsf{sk}, \mathsf{pk})$ and publishes $\mathsf{pk}$. The evaluation algorithm $\mathsf{Eval}(\mathsf{sk}, x)$ acts as follows:

1. Let $\widehat{Y} \leftarrow H(x)$ where $H$ maps arbitrary strings into $\mathbb{Z}^*_{\mathsf{pk}^2}$ (the domain of Paillier ciphertexts).
2. Decrypt $\widehat{Y}$ using $\mathsf{sk}$ to get the plaintext $y$.

3. Output $(y, Y \leftarrow y \cdot G, \pi)$, where $\pi$ is an *equality proof* of $(\widehat{Y}, Y)$: proving that $\widehat{Y}$ encrypts $\log Y$. The verification algorithm merely computes $\widehat{Y} := H(x)$ and verifies $\pi$.

Since Paillier is binding and hiding (for the latter, assuming DCRA holds), the security of the above eVRF is rather straightforward. Of course, there are some subtitles regarding the actual implementation of the above: what guarantees that $\mathsf{pk}$ is a valid Paillier public key? What if $\widehat{Y}$ is not a valid ciphertext (i.e., $\gcd(\widehat{Y}, \mathsf{pk}) > 1$)? But the only real obstacle is the equality proof used in the last step; efficient equality proofs require $y$ to be significantly smaller than $\mathsf{pk}$. While we can make $\mathsf{pk}$ to be sufficiently larger than $q \leftarrow |\mathbb{G}|$, clearly we cannot make $H(x)$ output an encryption of small values. The solution is surprisingly simple: $\mathsf{Eval}(\mathsf{sk}, x)$ *publishes* $o \leftarrow \lfloor y/q \rfloor$, and continues with $\widehat{Y}' \leftarrow \widehat{Y} - q \cdot \widehat{O}$, where $\widehat{O}$ is an encryption of $o$ (using a fixed randomness). Note that $\widehat{Y}'$ encrypts a small value, and thus standard equality proofs can be used to prove that $\widehat{Y}'$ encrypts $\log Y$. The crux of the above is that while $\mathsf{Eval}$ can cheat about the value of $o$, it is guaranteed that $y' = y \bmod q$, where $y'$ is the value encrypted in $\widehat{Y}'$ (so verifiability holds). In addition, while $o$ leaks information about $y$, it leaks *no information* about $y \bmod q$ which is all that we care about (so simulatability holds).

We present the eVRF construction in Section 6.1, using an ideal equality proof, and prove its security in Section 6.2. Efficient implementations of the ZK proofs used by the construction (including the aforementioned equality proof) are discussed in Section 6.3. Finally, in Section 6.3 we address the ZK-POK of the eVRF secret key required for the ideal/real security realization of the scheme.

### 6.1 The Construction

Let $(\mathsf{PaillierKeyGenEx}, \mathsf{PaillierEnc}, \mathsf{PaillierDecEx})$ be the standard key generation, encryption and decryption algorithm of the Paillier scheme, where $\mathsf{PaillierKeyGenEx}$ outputs also the factors of the public key, in addition to the public and secrete keys, and $\mathsf{PaillierDecEx}$ output also the randomness leading to the ciphertext.

In the following, we fix an ensemble of efficient additive cyclic groups $\{\mathbb{G}_\lambda\}$: there exists efficient algorithms for addition in the group, and for finding the specified generator $G_\lambda$. We make use of zero-knowledge proofs for the relations:

- $\mathcal{R}_{\mathsf{PaillierValidPK}} = \{(\mathsf{pk}, \mathsf{sk}) \colon \gcd(\mathsf{pk}, \mathsf{sk}) = 1 \wedge \mathsf{sk} = \phi(\mathsf{pk})\}$.
- $\mathcal{R}_{\mathsf{PaillierDlogEq}} = \left\{(\lambda, \mathsf{pk}, C, \widehat{C}), (c, r)) \colon (c \cdot G_\lambda = C) \wedge (\widehat{C} \in \mathbb{Z}^*_{\mathsf{pk}^2}) \wedge (\mathsf{PaillierEnc}_{\mathsf{pk}}(c; r) = \widehat{C})\right\}$.

**Construction 16 (Paillier based eVRF)** *The* Paillier-based eVRF *scheme with respect to function-family ensemble $\mathcal{H}$ mapping $\{0,1\}^*$ to $\mathbb{N}$, and non-interactive proofs $\Pi_{\mathsf{PaillierValidPK}} = (\mathsf{P}_{\mathsf{PaillierValidPK}}, \mathsf{V}_{\mathsf{PaillierValidPK}})$ and $\Pi_{\mathsf{PaillierDlogEq}} = (\mathsf{P}_{\mathsf{PaillierDlogEq}}, \mathsf{V}_{\mathsf{PaillierDlogEq}})$, is defined, for any $\lambda \in \mathbb{N}$ and $h \in \mathcal{H}_\lambda$, as follows.*

**Key generation.** $\mathsf{KeyGen}(1^\lambda)$*:*

1. $(\mathsf{pk}, \mathsf{sk}, p_1, p_2) \leftarrow\!\!\$\ \mathsf{PaillierKeyGenEx}(\lambda)$.
2. $\pi^{PK} \leftarrow \mathsf{P}_{\mathsf{PaillierValidPK}}(\mathsf{pk}, (p_1, p_2))$.
3. *Output* $(\mathsf{sk}, \mathsf{vk} \leftarrow (\mathsf{pk}, \pi^{PK}))$.

**Evaluation.** $\mathsf{Eval}^h(1^\lambda, k, x)$*:*

1. *Let* $\widehat{Y} \leftarrow h(x) \bmod \mathsf{pk}^2$. *Abort if* $\gcd(\widehat{Y}, \mathsf{pk}) \neq 1$.

2. *Let* $(y, r) \leftarrow \mathsf{PaillierDecEx}_{\mathsf{sk}}(\widehat{Y})$ *and* $Y \leftarrow y \cdot G_\lambda$.

3. *Let* $\pi \leftarrow \mathsf{P}_{\mathsf{PaillierDlogEq}}((\lambda, \mathsf{pk}, Y, \widehat{Y}), (y, r))$.

4. *Output* $(y, Y, \pi)$.

**Verfication.** $\mathsf{Verify}^h((1^\lambda, \mathsf{pk}, \pi^{PK}), x, Y, \pi)$*: Let* $\widehat{Y} \leftarrow g(x)$*. Accept if* (1) $\mathsf{pk} > q$,
(2) $\mathsf{V}_{\mathsf{PaillierValidPK}}(\mathsf{pk}, \pi^{PK})$, (3) $\mathsf{V}_{\mathsf{PaillierDlogEq}}((\lambda, \mathsf{pk}, Y, (h(x) \bmod \mathsf{pk}^2)), \pi)$.[12]

### 6.2 Security

Recall Definition 4 for the definition of (oracle-aided) eVRF. We assume without loss of generality that that for any $\mathsf{pk} \in \mathsf{Supp}(\mathsf{PaillierKeyGenEx}(\lambda)_1)$, it holds that $\mathsf{pk} \in [|\mathbb{G}_\lambda| \cdot 2^\lambda, 2^{c\lambda}]$ for some universal constant $c \in \mathbb{N}$, and let $\ell(\lambda) := (2c+1)\lambda$.

**Theorem 17.** *Assume that the DCRA assumption holds, and that* $\Pi_{\mathsf{PaillierValidPK}}$ *and* $\Pi_{\mathsf{PaillierDlogEq}}$ *are zero-knowledge proofs for* $\mathcal{R}_{\mathsf{PaillierValidPK}}$ *and* $\mathcal{R}_{\mathsf{PaillierDlogEq}}$ *respectively. Then Construction 16 is a secure eVRF with respect to domain/range ensemble* $\{(\{0,1\}^*, \mathbb{G}_\lambda)\}_{\lambda \in \mathbb{N}}$ *and function-family ensemble* $\left\{\mathcal{O}_{\{0,1\}^*, \{0,1\}^{\ell(\lambda)}}\right\}_{\lambda \in \mathbb{N}}$.

*Proof.* In the following, we fix the security parameter $\lambda$, let $q \leftarrow |\mathbb{G}_\lambda|$, and let $g(x) := h(x) \bmod \mathsf{pk}^2$. We omit $\lambda$ from the notation when clear from the context. Note that for any $x \in \{0,1\}^*$, the value of $\widehat{Y} \leftarrow g(x)$ is $2^{-\lambda}$ close to uniform over $\mathbb{Z}^*_{\mathsf{pk}^2}$. Since DCRA implies that factoring is hard, the above yields that for any PPT A:

$$\Pr_{h \leftarrow\$ \mathcal{H}_\lambda; (\mathsf{sk}, \mathsf{vk}) \leftarrow\$ \mathsf{KeyGen}(1^\lambda)} \left[ x \leftarrow\$ \mathsf{A}^h(\mathsf{sk}, \mathsf{vk}) \colon \mathsf{Eval}^h_{\mathsf{sk}}(x) = \perp \right] \leq \mathsf{negl}(\lambda) \tag{11}$$

letting $\perp$ indicate that Eval aborts. Hence, consistency holds with all but negligible probability. We continue by separately proving the security of $\mathsf{Eval}_1$ and $\mathsf{Eval}_2$.

**Security of** $\mathsf{Eval}_1$**.** Since $2^\ell \geq \mathsf{pk}^2 \cdot 2^\lambda$, the value of $g(x)$ is $2^{-\lambda}$-close to be uniform over $\mathbb{Z}^*_{\mathsf{pk}^2}$ (this holds, for any $x$ independently). Thus, the value of $y$ for $(y, \cdot) \leftarrow \mathsf{PaillierDecEx}_{\mathsf{sk}}(g(x))$, is $2^{-\lambda}$-close to uniform over $\mathbb{Z}_{\mathsf{pk}}$. Further, since (by assumption) $\mathsf{pk} \geq q \cdot 2^\lambda$, the value of $(y \bmod q)$ is $2^{1-\lambda}$-close to uniform over $\mathbb{Z}_q$. We conclude the proof by using the CPA security of the encryption scheme to show that this indistinguishably from uniform still holds when the distinguisher can query $h$ and $\mathsf{Eval}_1$.

Let D be a $t$-query efficient algorithm that presumably breaks pseudorandomness of the scheme. Assume for simplicity that D distinguishes between the hybrid in which all outputs are according to the function (hereafter, the *correct hybrid*), and the one in which the answer to one of the queries (whose indexed is sampled at random) is uniformly sampled (the *skewed hybrid*). We assume without loss of generality that before D queries $\mathsf{Eval}_1$ on $x$, it queries $h$ on $x$, and that it never makes the same query twice. Consider the following algorithm D' for winning the CPA game.

**Algorithm 18 (CPA breaker D')**
*Input:* pk.
*Operation:*

[12] The first two checks can be done once per verification key.

1. *Send* $m_0, m_1 \leftarrow\!\!\$\ \mathbb{Z}_{\mathsf{pk}}$ *to the challenger. Let* $\widehat{C}$ *be the challenger's answer (which either encrypts* $m_0$ *or* $m_1$*).*
2. *Call the ZK simulator of* $\Pi_{\mathcal{R}_{\mathsf{PaillierValidPK}}}$ *on input* $\mathsf{pk}$*, let* $\pi^{PK}$ *be its answer. Set* $\mathsf{vk} \leftarrow (\mathsf{pk}, \pi^{PK})$*.*
3. *Sample* $i^* \leftarrow\!\!\$\ [t]$*.*
4. *Start emulating* $\mathsf{D}(\mathsf{vk})$*, while answering its oracle calls as follows:*

   $h$ **calls.** *On the* $i$*'th query to* $h$*:*

   (a) *If* $i \neq i^*$*, sample* $y, r \leftarrow\!\!\$\ \mathbb{Z}_{\mathsf{pk}}$ *and let* $\widehat{Y} \leftarrow \mathsf{PaillierEnc}_{\mathsf{pk}}(y; r)$*.*
   *Else, let* $\widehat{Y} \leftarrow \widehat{C}$ *and* $y \leftarrow y_0$*.*

   (b) *Store* $(\widehat{Y}, y)$*.*

   (c) *Return* $c \leftarrow\!\!\$\ \{0,1\}^\ell$ *conditioned on* $c = \widehat{Y} \bmod \mathsf{pk}^2$*.*

   $\mathsf{Eval}_1$ **calls.** *On query* $x$*:*

   (a) *Let* $\widehat{Y} \leftarrow g(x) = h(x) \bmod \mathsf{pk}^2$ *(according to the answers produced above), and let* $y$ *be the value stored for* $\widehat{Y}$*.*

   (b) *Return* $y$*.*
5. *Output* $\mathsf{D}$*'s output.*

Note that if $\widehat{C}$ encrypts $y_1$ and $\mathsf{Eval}_1$ was called on the $i^*$ query of $h$, the above execution is according to the skewed hybrid. Otherwise, it is according to the correct hybrid. It follows that $\mathsf{D}'$ breaks the CPA security with probability $1/t^2$ times the probability that $\mathsf{D}$ breaks the pseudorandomness of $\mathsf{Eval}_1$. (One $1/t$ factor is for the hybrid argument, and the second $1/t$ factor is for the guess of $i^*$.)

**Security of** $\mathsf{Eval}_2$**.**

- Correctness. Immediate by Equation (11).
- Pseudorandomness. Immediately follows from that of $\mathsf{Eval}_1$ and the consistency guarantee.
- Simulatability. Immediately follows by the zero-knowledge of $\Pi_{\mathsf{PaillierDlogEq}}$. That is, $\mathsf{Sim}(\mathsf{vk}, x, Y)$ acts as follows:
  1. Let $\widehat{Y} \leftarrow g(x)$.
  2. Invoke the guaranteed zero-knowledge of $\Pi_{\mathsf{PaillierDlogEq}}$ on $(\widehat{Y}, Y)$ to generate a (fake) proof $\pi$.
  3. Output $(Y, \pi)$.
- Verifiability. Easily follows from the binding of the Paillier encryption and the soundness of the zero-knowledge proofs; the soundness of $\Pi_{\mathsf{PaillierValidPK}}$ yields that the probability of accept on a verification key $\mathsf{vk} = (\mathsf{pk}, \cdot)$ with a non well-formed $\mathsf{pk}$ (i.e., $\gcd(\mathsf{pk}, \phi(\mathsf{pk})) \neq 1$), is negligible. So in the following we fix a well-formed $\mathsf{pk}$. Fix $h$ and $x$, and let $\widehat{Y} \leftarrow g(x) = h(x) \bmod \mathsf{pk}^2$. Since the verifier aborts if $\gcd(\widehat{Y}, \mathsf{pk}) \neq 1$, we assume this is not the case. Since $\mathsf{pk}$ is well formed, there exists a unique $(y, r) \in \mathbb{Z}_n \times \mathbb{Z}_n^*$ with $\mathsf{PaillierEnc}_{\mathsf{pk}}(y; r) = \widehat{Y}$. Thus, the soundness of $\Pi_{\mathsf{PaillierDlogEq}}$ yields that the probability of efficiently finding a pair $(Y', \pi)$ with $Y' \neq y \cdot G$ that makes the verifier accept, is negligible.

□

### 6.3 Implementing the Zero-Knowledge Proofs

$\mathcal{R}_{\mathsf{PaillierValidPK}}$. We can use the efficient proof of [21, Section 3.2].

$\mathcal{R}_{\mathsf{PaillierDlogEq}}$. Efficient proofs for this relation, known as equality proofs, require the plaintext hidden in the Paillier ciphertext to be much smaller than $\mathsf{pk}$, cf., [26,5]. Specifically, it is needed that for the plaintext $y$ it holds that $y \leq \mathsf{pk}/2^{c\lambda}$, where $c$ is typically 2. To use such proofs, we employ the following modifications:

**Key generation.** The call to $\mathsf{KeyGen}(1^\lambda)$ is adjusted to ensure $\mathsf{pk} > q \cdot 2^{c\lambda}$.
**Evaluation.** $\mathsf{Eval}^h(k, x)$:

1-4. Same as in Construction 16.
5. Let $y' \leftarrow y \bmod q$, $o \leftarrow \lfloor y/q \rfloor$, and $\widehat{Y}' \leftarrow \widehat{Y} - \mathsf{PaillierEnc}_{\mathsf{pk}}(o \cdot q; 1)$ (i.e., "reduce" $o \cdot q$ from $\widehat{Y}$).
6. Let $\pi \leftarrow \mathsf{P}_{\mathsf{PaillierDlogEq}}((\mathsf{pk}, Y, \widehat{Y}'), (y', r))$.
7. Output $(y, Y, \pi' = (\pi, o))$.

**Verfication.** $\mathsf{Verify}^h((\mathsf{pk}, \pi^{PK}), x, Y, \pi') = (\pi, o))$:

1-2. Same as in Construction 16.
3. $\mathsf{V}_{\mathsf{PaillierDlogEq}}((\mathsf{pk}, Y, g(x) - \mathsf{PaillierEnc}_{\mathsf{pk}}(o \cdot q; 1)), \pi)$.

That is, the value of $y$ is replaced with a smaller $y' \in \mathbb{Z}_q$ with $y' = y \bmod q$, and the proof continues using an encryption $\widehat{Y}'$ of $y'$. While the evaluater has a freedom in choosing the value of $\widehat{Y}'$, and the corresponding plaintext $y'$, it has no control on $y' \bmod q$. So security holds as in the original protocol. Since $y'$ is small and $\mathsf{pk}$ is large enough, we can use a more relax equality proof.

**Range proofs.** Equality proofs of the type mentioned above are using *range proofs* for proving that the encrypted plaintext is small. There are several approaches to implement range proof for Paillier ciphertexts. The basic approach uses cut-and-chose, which can be carried over the Paillier domain, and requires the prover to send an encryption of a random value. Alternatively, it can be carried over *Pedersen commitments* over *known* (prime) order groups, and requires the prover to send a commitment of random message. In both cases, the basic protocol has soundness error 1/2, and thus the final protocol requires many repetitions.

Range proofs of much better efficiency can be based on Pedersen commitments over RSA groups, typically referred to as *integer commitments*. Unfortunately, the security of these proofs require the factorization of the RSA to be *unknown* to the prover. In our setting, it means that the RSA group cannot be sampled by the key-generation or evaluation algorithms. This obstacle can be overcome by sampling the RSA group in a trusted setup phase. Note that the same RSA group can be used by all instantiations of the eVRF, making it a reasonable solution in some settings.[13]

**Knowledge of Secret Key.** Uplifting the security of the scheme from a game-based one (Definition 4) to realizing the ideal functionality (Definition 5), requires a ZK-POK proof for the relation $\{((\mathsf{pk}, \lambda), \mathsf{sk}) \colon (\mathsf{pk}, \mathsf{sk}, \cdot) \in \mathsf{Supp}(\mathsf{PaillierKeyGenEx}(1^\lambda))\}$. Since the Paillier secret key is easily deduce from the factors of the public key, one can use the ZK-POK proof from [13, Section 6.3.1] for the relation $\mathcal{R}_{\mathsf{PaillierSK}} = \{(\mathsf{pk}, (p, q)) \colon p \cdot q = \mathsf{pk}\}$.

[13] Integer commitments also require generators of the quadratic residuosity group, which can also sampled in the trusted setup phase.

**Running time.** The cost of proving $\mathcal{R}_{\mathsf{PaillierValidPK}}$ (with the protocol of [21, Section 3.2] with Fiat-Shamir) is 15ms, and the cost of verifying is 17ms (this is only carried out during initialization). The cost of proving $\mathcal{R}_{\mathsf{PaillierDlogEq}}$ using a range proof over a prime-order group (and therefore not requiring any trusted setup) is 360ms for proving and 151ms for verifying. The cost when using range proofs over integer commitments (requiring a trusted setup) is over an order of magnitude lower. The only other operations required are a Paillier decryption and an elliptic curve multiplication, requiring a few milliseconds only.

## 7 Conclusions and Open Problems

In this paper, we have introduced a new primitive called an exponent VRF (eVRF), and shown that it has many applications in the field of threshold cryptography and signing. In particular, it enables us to achieve one-round simulatable key generation, two-round signing for Schnorr (multiparty) and ECDSA (two-party), and it provides a hierarchical key derivation method like BIP032 with additional properties like MPC friendliness and public verifiability. We also provided constructions under both the DDH and Paillier (DCRA) assumptions.

Our work leaves open a number of interesting questions. An important open question raised by this work is the construction of an efficient key homomorphic eVRF, namely an eVRF that satisfies

$$\mathsf{Eval}_1(k_1 + k_2, x) \cdot G = \mathsf{Eval}_1(k_1, x) \cdot G + \mathsf{Eval}_1(k_2, x) \cdot G$$

where $G$ is a generator of a standard cryptographic group used for Schnorr signatures. This will enable *deterministic* two round threshold Schnorr signing, by constructing a threshold eVRF itself so that any subset $\mathcal{Q}$ of the parties will compute the same $\mathsf{Eval}$ result on the same message. In addition, it will enable threshold Schnorr signing without knowing the set of parties ahead of time. Finally, it will enable the parties to refresh their secrets and achieve (static) proactive security.

One possible approach to constructing a key homomorphic eVRF is to explore building an efficient eVRF from a lattice-based random oracle PRF, described in [6], whose security is based on the lattice with rounding problem (LWR). This PRF is almost key homomorphic, which is sufficient for the applications in Section 4. One would build an eVRF by encoding the output of this PRF in the exponent of another group, as we did in Sections 5 and 6. The challenge then is to devise an efficient non-interactive zero-knowledge proof that the PRF was evaluated correctly.

Another important open question is to construct a simulatable two-round *multiparty* protocol for ECDSA (our ECDSA protocol is only for two parties).

**Acknowledgments.** The first author was supported by NSF, DARPA, the Simons Foundation, and NTT Research. Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

## References

1. Attema, T., Fehr, S., Klooß, M.: Fiat-shamir transformation of multi-round interactive proofs. In: Kiltz, E., Vaikuntanathan, V. (eds.) TCC 2022, Part I. LNCS, vol. 13747, pp. 113–142. Springer, Heidelberg, Germany, Chicago, IL, USA (Nov 7–10, 2022). https://doi.org/10.1007/978-3-031-22318-1_5
2. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. Cryptology ePrint Archive, Report 2011/368 (2011), https://eprint.iacr.org/2011/368
3. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. Journal of Cryptographic Engineering **2**(2), 77–89 (Sep 2012). https://doi.org/10.1007/s13389-012-0027-1

4. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: Goldwasser, S. (ed.) ITCS 2012. pp. 326–349. ACM, Cambridge, MA, USA (Jan 8–10, 2012). https://doi.org/10.1145/2090236.2090263
5. Blokh, C., Makriyannis, N., Peled, U.: Efficient asymmetric threshold ECDSA for mpc-based cold storage. IACR Cryptol. ePrint Arch. p. 1296 (2022)
6. Boneh, D., Lewi, K., Montgomery, H.W., Raghunathan, A.: Key homomorphic PRFs and their applications. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 410–428. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2013). https://doi.org/10.1007/978-3-642-40041-4_23
7. Boneh, D., Shoup, V.: A graduate course in applied cryptography (version 0.6). Cambridge University Press (2023), cryptobook.us
8. Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 327–357. Springer, Heidelberg, Germany, Vienna, Austria (May 8–12, 2016). https://doi.org/10.1007/978-3-662-49896-5_12
9. Bünz, B.: Improving the privacy, scalability, and ecological impact of blockchains. Ph.D. thesis, Stanford University (2023)
10. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy. pp. 315–334. IEEE Computer Society Press, San Francisco, CA, USA (May 21–23, 2018). https://doi.org/10.1109/SP.2018.00020
11. Canetti, R.: Security and composition of multiparty cryptographic protocols. Journal of Cryptology **13**(1), 143–202 (Jan 2000). https://doi.org/10.1007/s001459910006
12. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press, Las Vegas, NV, USA (Oct 14–17, 2001). https://doi.org/10.1109/SFCS.2001.959888
13. Canetti, R., Gennaro, R., Goldfeder, S., Makriyannis, N., Peled, U.: UC non-interactive, proactive, threshold ECDSA with identifiable aborts. Cryptology ePrint Archive, Report 2021/060 (2021), https://eprint.iacr.org/2021/060
14. Chen, Y.H., Lindell, Y.: Feldman's verifiable secret sharing for a dishonest majority. Cryptology ePrint Archive, Paper 2024/031 (2024), https://eprint.iacr.org/2024/031, https://eprint.iacr.org/2024/031
15. Doerner, J., Kondi, Y., Lee, E., abhi shelat: Threshold ECDSA in three rounds. Cryptology ePrint Archive, Paper 2023/765 (2023), https://eprint.iacr.org/2023/765, https://eprint.iacr.org/2023/765
16. Doerner, J., Kondi, Y., Lee, E., shelat, a.: Secure two-party threshold ECDSA from ECDSA assumptions. Cryptology ePrint Archive, Report 2018/499 (2018), https://eprint.iacr.org/2018/499
17. Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: 28th FOCS. pp. 427–437. IEEE Computer Society Press, Los Angeles, CA, USA (Oct 12–14, 1987). https://doi.org/10.1109/SFCS.1987.4
18. Fischlin, M.: Communication-efficient non-interactive proofs of knowledge with online extractors. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 152–168. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2005). https://doi.org/10.1007/11535218_10
19. Garg, S., Srinivasan, A.: Two-round multiparty secure computation from minimal assumptions. J. ACM **69**(5) (2022). https://doi.org/10.1145/3566048, https://doi.org/10.1145/3566048
20. Garillot, F., Kondi, Y., Mohassel, P., Nikolaenko, V.: Threshold Schnorr with stateless deterministic signing from standard assumptions. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part I. LNCS, vol. 12825, pp. 127–156. Springer, Heidelberg, Germany, Virtual Event (Aug 16–20, 2021). https://doi.org/10.1007/978-3-030-84242-0_6
21. Goldberg, S., Reyzin, L., Sagga, O., Baldimtsi, F.: Efficient noninteractive certification of RSA moduli and beyond. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part III. LNCS, vol. 11923, pp. 700–727. Springer, Heidelberg, Germany, Kobe, Japan (Dec 8–12, 2019). https://doi.org/10.1007/978-3-030-34618-8_24
22. Goldreich, O.: Foundations of Cryptography: Basic Applications, vol. 2. Cambridge University Press, Cambridge, UK (2004)
23. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. Journal of the ACM **33**(4), 792–807 (Oct 1986)
24. Komlo, C., Goldberg, I.: FROST: Flexible round-optimized Schnorr threshold signatures. In: Dunkelman, O., Jr., M.J.J., O'Flynn, C. (eds.) SAC 2020. LNCS, vol. 12804, pp. 34–65. Springer, Heidelberg, Germany, Halifax, NS, Canada (Virtual Event) (Oct 21-23, 2020). https://doi.org/10.1007/978-3-030-81652-0_2
25. Kushilevitz, E., Lindell, Y., Rabin, T.: Information-theoretically secure protocols and security under composition. In: Kleinberg, J.M. (ed.) 38th ACM STOC. pp. 109–118. ACM Press, Seattle, WA, USA (May 21–23, 2006). https://doi.org/10.1145/1132516.1132532

26. Lindell, Y.: Fast secure two-party ECDSA signing. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 613–644. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2017). https://doi.org/10.1007/978-3-319-63715-0_21
27. Lindell, Y.: Simple three-round multiparty schnorr signing with full simulatability. Cryptology ePrint Archive, Report 2022/374 (2022), https://eprint.iacr.org/2022/374
28. Makriyannis, N.: On the classic protocol for MPC schnorr signatures. Cryptology ePrint Archive, Report 2022/1332 (2022), https://eprint.iacr.org/2022/1332
29. Makriyannis, N., Yomtov, O., Galansky, A.: Practical key-extraction attacks in leading mpc wallets. Cryptology ePrint Archive, Paper 2023/1234 (2023), https://eprint.iacr.org/2023/1234, https://eprint.iacr.org/2023/1234
30. Micali, S., Rabin, M.O., Vadhan, S.P.: Verifiable random functions. In: 40th FOCS. pp. 120–130. IEEE Computer Society Press, New York, NY, USA (Oct 17–19, 1999). https://doi.org/10.1109/SFFCS.1999.814584
31. Naor, M., Pinkas, B., Reingold, O.: Distributed pseudo-random functions and KDCs. In: Stern, J. (ed.) EUROCRYPT'99. LNCS, vol. 1592, pp. 327–346. Springer, Heidelberg, Germany, Prague, Czech Republic (May 2–6, 1999). https://doi.org/10.1007/3-540-48910-X_23
32. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT'99. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg, Germany, Prague, Czech Republic (May 2–6, 1999). https://doi.org/10.1007/3-540-48910-X_16
33. Palatinus, M., Rusnak, P.: Multi-account hierarchy for deterministic wallets (2014), https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki
34. Papadopoulos, D., Wessels, D., Huque, S., Naor, M., Včelák, J., Reyzin, L., Goldberg, S.: Making NSEC5 practical for DNSSEC. Cryptology ePrint Archive, Report 2017/099 (2017), https://eprint.iacr.org/2017/099
35. Pippenger, N.: On the evaluation of powers and monomials. SIAM Journal on Computing **9**(2), 230–250 (1980)
36. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of Cryptology **4**(3), 161–174 (Jan 1991). https://doi.org/10.1007/BF00196725
37. Silverman, J.H., Stange, K.E.: Amicable pairs and aliquot cycles for elliptic curves. Experimental Mathematics **20**(3) (2011). https://doi.org/10.1080/10586458.2011, link
38. Wuille, P.: Hierarchical deterministic wallets (2012), https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki

# A The R1CS matrices $A, B, C$ used in Section 5.1

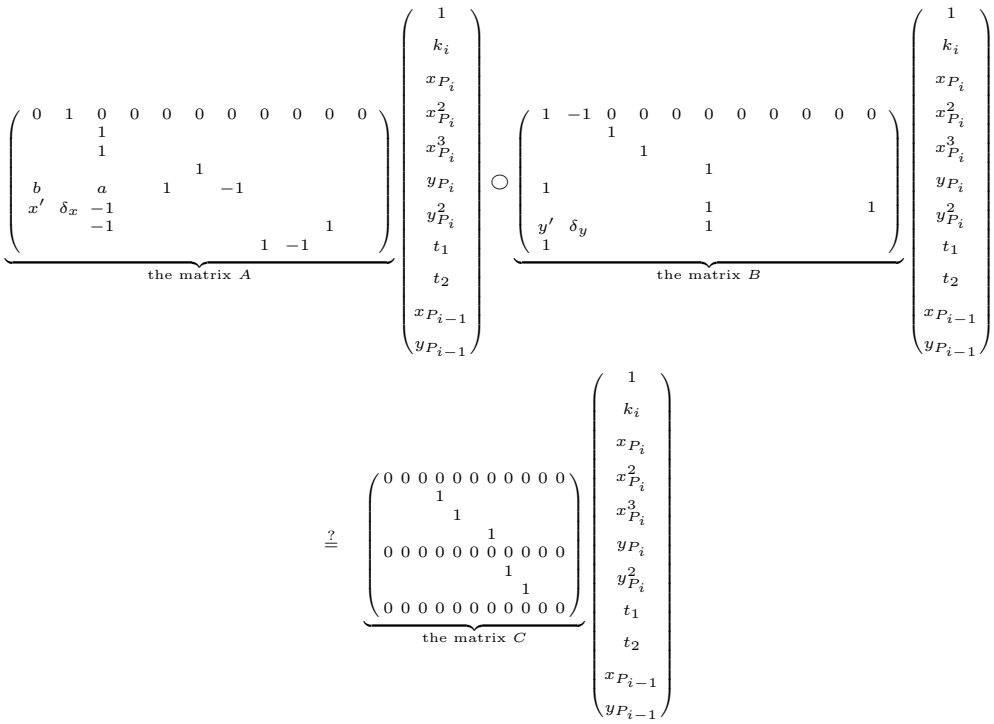


**Fig. 1.** The R1CS matrices $A, B, C$ used in Section 5.1 for the $i$'th block of checks for some $i \in [\ell]$. Empty cells are set to 0. The first row confirms that $k_i \times (1 - k_i) = 0$. The second row confirms that $x_{P_i} \times x_{P_i} = x^2_{P_i}$. The third row confirms that $x_{P_i} \times x^2_{P_i} = x^3_{P_i}$. The fourth row confirms that $y_{P_i} \times y_{P_i} = y^2_{P_i}$. The fifth row confirms that $x^3_{P_i} + ax_{P_i} + b - y^2_{P_i} = 0$. The sixth row confirms that $(\delta_x k_i + x' - x_{P_i}) \times (y_{P_i} + y_{P_{i-1}}) = t_1$. The seventh row confirms that $(x_{P_{i-1}} - x_{P_i}) \times (\delta_y k_i + y' + y_{P_i}) = t_2$. The final row confirms that $t_1 - t_2 = 0$. The prover and verifier compute the values $\delta_x, \delta_y, x', y'$ using (8). Note that the 5th and 8th rows check a linear relation and can be combined into a single row by taking a linear combination using verifier randomness. The same holds for the second and third rows.