Ring Verifiable Random Functions and Zero-Knowledge Continuations

Jeffrey Burdges[1], Oana Ciobotaru[2], Handan Kılınç Alper[1], Alistair Stewart[1], and Sergey Vasilyev[1]

[1] Web 3.0 Foundation
[2] OpenZeppelin

October 19, 2023

Abstract. We introduce a new cryptographic primitive, named *ring verifiable random function (ring VRF)*. Ring VRF combines properties of VRF and ring signatures, offering verifiable unique, pseudorandom outputs while ensuring anonymity of the output and message authentication. We design its security in the universal composability (UC) framework and construct two protocols secure in our model. We also formalize a new notion of *zero-knowledge (ZK) continuations* allowing for the reusability of proofs by randomizing and enhancing the efficiency of one of our ring VRF schemes. We instantiate this notion with our protocol SpecialG which allows a prover to reprove a statement in a constant time and be unlikable to the previous proof(s).

1 Introduction

We introduce a novel cryptographic primitive called *a ring verifiable random function (ring VRF)*. Ring VRF operates in a manner akin to both VRF [37] and ring signatures [5,36,35,3,14], leveraging the properties of uniqueness, pseudorandomness, and anonymity. In ring VRF, a user can generate a ring VRF output, which is a *unique* pseudorandom number, with their key and input similar to VRF. They also sign the input and any message (e.g., auxiliary data) with a set of public keys (ring) including their key, similar to the ring signatures. The ring signature assures that the ring VRF output is the unique output of the input generated with one of the public keys and the same key signs also the message. The verification process does not reveal the signer's key except that their key is in the public key set.

The distinctive properties of ring VRF such as pseudorandomness, anonymity and uniqueness offer an efficient alternative for anonymous access control systems. Imagine an identity system where a user registers with their public key. Assuming that the system maintains a fixed input for a given service (e.g., urls) and provides a public commitment of the registered public keys, a registered user can create a ring VRF output using the fixed input and their key, which serves as their pseudonym. The user can then use this pseudonym as an identity while accessing a service provided by the system. At the same time, they can prove that their pseudonyms are legitimate all without revealing their true identity. Namely, they generate a ring VRF signature which shows that their pseudonym is associated with one of the registered users. In this way, the identity system protects the user's privacy. Moreover, the system is protected against the Sybil behaviour, as the ring VRF protocol ensures that a user can produce only one pseudonym per input. This protection enables the system to ban certain pseudonyms in cases of abusive behaviours. Thus, the abusive user loses the access since they cannot generate another legitimate pseudonym for this particular service. In current anonymous systems, user accountability is primarily addressed through two main approaches: (1) allowing users to authenticate for a fixed duration [43,38,8], and (2) incorporating mechanisms for privacy revocation administered by a central authority [9,33,20,44], or through privacy revocation using anonymous committees [7,17]. In contrast, Ring VRF offers a straightforward, efficient and succinct solution for user accountability when compared to existing methods as it neither imposes limitations on user behaviours nor necessitates the involvement of central authorities or anonymous committees to revoke the privacy of a malicious user. In addition to facilitate anonymous authentication, ring VRF serves as a potent tool for the concept of proof-of-personhood (PoP) [19,6,18] to establish a connection between the physical entities and virtual identities by preserving the accountability and anonymity of the entity.

Unique ring signature (URS) schemes [21] aim to address similar challenges as ring VRF in the context of anonymous identity applications. Both generate a unique identifier within the ring signature for each input, which corresponds to the ring VRF output in our case. Unlike ring VRF, where a party can sign any message with a ring VRF signature, unique ring signature schemes do not include the capability to sign such messages. Therefore, leveraging these identifiers for practical authentication, such as in a TLS session, is not straightforward. Beyond this, we demand from a ring VRF output to be a pseudorandom even if the signer's key is maliciously generated. This property distinguishes it from unique ring signatures. Although this property may not find immediate use in the identity applications we mentioned, it holds critical significance in applications that grant privileges to parties based on specific criteria associated with their ring VRF output, such as leader elections or lotteries. For instance, blockchain protocols often select leaders to produce a block based on the VRF output of a party, with parties having a VRF output below a certain threshold being chosen as leaders [16,1]. Since ring VRF provides the same pseudorandomness property required in these leader election mechanisms, a ring VRF scheme can potentially replace VRF in these protocols to provide also anonymity to leaders even after they produce their blocks. We remark that VRF cannot provide this level of anonymity, as verifying the correctness of the VRF output of a leader, which is necessary to verify the block of the leader, requires knowledge of the leader's public key.

We design two efficient ring VRF protocols that can be applied to real-world scenarios. In simple terms, our ring VRF signatures has components dedicated to verifying the output and confirming the key membership. Some scenarios require the generation of multiple ring VRF signatures for different inputs for *the same ring*. In these scenarios, since the ring does not change only the output changes, an optimized approach to generate a new signature given another signature generated for the same ring would be as follows: generate a new component only for the aspects directly associated with the correctness of the ring VRF output and rerandomize the relevant component of a prior signature indicating the existence of the signing key in the ring. This optimized solution at the same time should preserve both verifiability and anonymity of the optimized signature. To this end, we introduce a new notion called *zero-knowledge continuations*. It provides a way to efficiently prove a statement with a simple transformation of an existing proof of the same statement. After this transformation the new proof remains unlinkable to the other proofs.

In short, our contributions in this paper are as follows:

– We formally define the security of a ring VRF in the universal composability (UC) model. For this, we construct a functionality $\mathcal{F}_{\mathsf{rvrf}}$ and verify the security properties that $\mathcal{F}_{\mathsf{rvrf}}$ provides.
– We introduce a new notion called zero-knowledge (ZK) continuations which defines the transformation of a valid proof into another valid and unlinkable proof of the same statement through efficient operations. Essentially, this allows a prover to generate an initially costly proof and subsequently reuse it by simply rerandomizing it, while maintaining unlinkability with other proofs.
– We construct two distinct ring VRF protocols. The first protocol is designed to be utilized with a non-interactive zero-knowledge (NIZK) proving system with our specific relations. The second protocol is more specialized, allowing instantiation with any zero-knowledge continuations. The latter offers an efficient solution for ring VRF applications that necessitate the generation of multiple signatures for the same ring. We show that both of our protocols are UC-secure.
– We construct a protocol called $\mathsf{SpecialG}$ which is a simple transformation of any Groth16 proof into a new proof by deploying the rerandomization idea of LegoSNARK ccGro16 [10]. We show that $\mathsf{SpecialG}$ is a zero-knowledge continuation, making it suitable for deployment in instantiating

our second protocol. $\mathsf{SpecialG}$'s reproving time is reduced to constant after running once a linear time proving algorithm.

1.1 Ring VRF Overview

As a beginning, we introduce a ring VRF interface, give a straightforward unamortised ring VRF protocol realising the desired security properties, and give some intuition for our later amortization technique. Similar to VRF [37], a ring VRF construction needs:

- $\mathsf{rVRF.KeyGen}$ outputs secret and public keys $(\mathsf{sk}, \mathsf{pk})$.
- $\mathsf{rVRF.Eval}(\mathsf{sk}, \mathsf{in}) \mapsto \mathsf{out}$: *deterministically* computes the VRF output out from a secret key sk and an input in.

We demand a pseudorandomness property from the output of Eval for all sk.

In contrast to VRF, a ring VRF scheme has the following algorithms operating directly upon set of public keys ring:

- $\mathsf{rVRF.Sign}(\mathsf{sk}, \mathsf{ring}, \mathsf{in}, \mathsf{ad}) \mapsto \sigma$: returns a ring VRF signature σ which is a proof for the ring VRF output of in as well as a signature signing ad.
- $\mathsf{rVRF.Verify}(\mathsf{ring}, \mathsf{in}, \mathsf{ad}, \sigma) \mapsto \mathsf{out} \vee \perp$: returns either an output out or else failure $\perp$. It returns out if σ signs ad with one of the keys in ring and out is the ring VRF output of in and the same key that signs ad.

Ring VRF protocols deviate from VRF protocols in that they do not need the public key of the signer during the verification process. Instead, they use a set of public keys including the signer's key, like ring signatures. However, ring VRF protocols distinguish themselves from ring signatures in the verification process as well in which the unique ring VRF output of the signer is revealed if the signature is successfully verified with ring. In essence, a verified ring VRF signature of an input in actually proves that out is the evaluation output of in generated by the signer's key.

We want to achieve anonymity in ring VRF protocols meaning that the verifier learns nothing about the signer except that the evaluation value of the signed input in is out and the signer's public key is in ring. An intuitive ring VRF protocol could be instantiated by making $\mathsf{rVRF.Eval}$ a pseudorandom function, and using a NIZK protocol NIZK where $\mathsf{rVRF.Sign}$ runs the proving algorithm and $\mathsf{rVRF.Verify}$ runs the verification algorithm of NIZK for a relation consisting of statement and witness pairs as follows:

$$\mathcal{R}_{\mathsf{rvrf}} = \left\{ (\mathsf{out}, \mathsf{in}, \mathsf{ring}); (\mathsf{sk}, \mathsf{pk}) \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{rVRF.KeyGen}, \\ \mathsf{pk} \in \mathsf{ring} \\ \mathsf{out} = \mathsf{rVRF.Eval}(\mathsf{sk}, \mathsf{in}) \end{array} \right\}$$

The zero-knowledge property of the NIZK ensures that our verifier learns nothing about the specific signer, except that their key is in the ring and maps in to out. Importantly, pseudorandomness also says that out is an anonymous identity for the specific signer, but only within the context of in. We ignore ad in $\mathcal{R}_{\mathsf{rvrf}}$ for now just for the sake of simplicity. Otherwise, we note that it is imperative to incorporate $\mathsf{rVRF.Sign}$ and $\mathsf{rVRF.Verify}$ to sign associated data ad.

If one used the ring VRF interface described above, then one needs time $O(|\mathsf{ring}|)$ in $\mathsf{rVRF.Sign}$ and $\mathsf{rVRF.Verify}$ merely to read their ring argument, which severely limits applications. Instead, we replace ring with a commitment to the ring such as Merkle tree root and run asymptotically faster. Therefore, we introduce the following algorithms for rVRF.

- $\mathsf{rVRF.CommitRing} : (\mathsf{ring}, \mathsf{pk}) \mapsto (\mathsf{comring}, \mathsf{opring})$ returns a commitment for a set ring of public keys, and the opening opring if $\mathsf{pk} \in \mathsf{ring}$ as well.

- $\mathsf{rVRF.OpenRing} : (\mathsf{comring}, \mathsf{opring}) \mapsto \mathsf{pk} \vee \perp$ returns a public key pk, provided opring correctly opens the ring commitment $\mathsf{comring}$, or failure $\perp$ otherwise.

Together with these algorithms, we can replace the membership condition $\mathsf{pk} \in \mathsf{ring}$ in $\mathcal{R}_{\mathsf{rvrf}}$ by the opening condition $\mathsf{pk} = \mathsf{rVRF.OpenRing}(\mathsf{comring}, \mathsf{opring})$ and replace ring in the statement with $\mathsf{comring}$ and add opring to the witness.

Our Approach: Although an asymptotic improvement with $\mathsf{rVRF.CommitRing}$, the intuitive scheme can be computationally expensive to prove the evaluation value together with the membership condition. Therefore, in our first ring VRF protocol, we divide the relation into two relations. The first relation $\mathcal{R}_{\mathtt{eval}}$ is designed to show the validity of the evaluation value with a proof that can be efficiently generated using discrete logarithm equality proofs. We integrate ad in this proof so that the proof serves as a signature of ad signed by the same key used to generate the ring VRF output. The second relation $\mathcal{R}_{\mathtt{ring}}$ is designed to show that the key used in the evaluation and signing ad is in the ring. Its statement therefore has one part of the evaluation proof which is the Pedersen commitment to the secret key in order to relate the key in $\mathcal{R}_{\mathtt{ring}}$ and $\mathcal{R}_{\mathtt{eval}}$.

The most computationally expensive part of our first protocol is generating a proof for $\mathcal{R}_{\mathtt{ring}}$ during signing. Therefore, we deploy a further optimization on this and design the second protocol. For this, we consider optimising the cases where a party generates ring VRF signatures of different inputs for the *same ring*. In this case, actually, the main change in the new signature is caused by the cheapest part of the signing process which generates proof for the ring VRF output and signature for ad because the input changes so the evaluation value changes. Consequently, it raises the question of why a party should re-execute the ring membership proving. In light of this, we deploy in the second protocol our new notion ZK continuation that allows us to generate proofs for $\mathcal{R}_{\mathsf{rvrf}}$ by reusing a previously generated proof for ring membership with simple operations. In a nutshell, if a party once generates a ring VRF signature for ring in our second scheme, this signature has a Pedersen commitment to the secret key sk i.e., $\mathsf{compk} = \mathsf{sk}G + \mathsf{b}K$ as a part of the proof for the output similar to our first scheme. When this party generates another signature for a different input with ring, they (re)randomize the existing proof for the ring membership with a random number b' rather than running the proving algorithm for the ring membership from scratch. Then, they generate a new proof for the ring VRF output by running the proving algorithm for the ring VRF output by setting the new Pedersen commitment to the secret key with $\mathsf{compk}' = \mathsf{sk}G + \mathsf{b}'K$ which is the new Pedersen commitment to the secret key generated with the same randomness b' used in ring membership.

1.2 Related Works

Security Models: The unique ring signature framework [21] is the closest model to our ring VRF framework particularly in terms of the presence of a deterministic component known as the unique identifier for the signed message. This identifier remains constant for the same signed message even when the ring changes. Essentially, the unique identifier in the unique ring signature model and the ring VRF evaluation value function equivalently in both models. However, a fundamental distinction lies in the treatment of this identifier. In our ring VRF model, we impose the requirement of pseudorandomness, as defined in [2,16], on this unique identifier, even in the case of malicious parties. This requirement is crucial for applications such as lotteries or leader elections where the unique identifier plays a privileged or reward-based role based on predefined conditions. Another definitional difference is that a ring VRF signature not only prove the correctness of the evaluation value of an input but also signs an auxiliary data independent from the input. This property is needed for anonymous access mechanisms to prevent replay attacks because auxiliary data can be used to effectively bind the ring VRF signature to e.g., a TLS session. The signature size of unique ring signature schemes scales either linearly [21,22] or logarithmically [42,39] with the size of the ring. In contrast, our ring VRF constructions maintain a constant signature size while providing stronger security guarantees. Also our signing and verification algorithms show better asymptotic scalability compared to existing unique ring signatures because they operate with a constant-size commitment to the ring.

Other related models are linkable ring signature [35,34] and traceable ring signature [25,24]. Linkable ring signatures allows a third party to link whether two ring signatures of two inputs are signed by the same party in the same ring without revealing the identity. This type of linkability property is valuable in applications that impose restrictions on authentications, such as preventing double spending or multiple voting. Akin to ring VRF and unique ring signatures, if a signer signs the same message twice for the same ring and issuer, it becomes evident that both signatures are produced by the same party, although the specific party's identity remains secret. Both ring VRF and unique ring signature schemes have this property in a single context through the unique identifier for each party. Differently than ring VRF, traceable ring signatures disclose the identity of the signer when the signer generates two signatures for two different inputs within the same ring and from the same issuer.

Another related informal design is Semaphore [31], which also provides a "nullifier", unique per identity and context but anonymous, (akin to a ring VRF output in our formalism) along with a signature on a message. However, the security properties of Semaphore are not fully formalized, and our constructions distinguishes themselves by offering more efficient proving times and the potential for proof reuse.

Anonymous VRF [45] is a special type of VRF designed to enable verification of the VRF output without dependence on the party's key. Differently than ring VRF, the verification is executed with another public key which is generated from the public key of the party. A crucial distinction lies in their uniqueness definitions, as anonymous VRFs ensure the uniqueness of VRF outputs for each (updated) public key and input. Consequently, anonymous VRFs are not suitable for identity applications where the VRF output serves as a unique and anonymous identifier, as each updated public key generates a different VRF output. Another notable difference is related to the pseudorandomness definition, which does not guarantee pseudorandomness even when the key belongs to a malicious party. This limitation can pose challenges in applications like consensus mechanisms as described in [45], making their use potentially infeasible.
Commit and Prove SNARKs: ZK Continuations are an example of the commit and prove approach [10], linking in a way similar to the ccGroth16 construction from LegoSNARK [10]. Our work extends this concept by formalizing the reuse of previously generated proofs through simple transformations while maintaining the zero-knowledge property. Our protocol $\mathsf{SpecialG}$ is very similar to the ccGroth16 construction from LegoSNARK [10] with the additional feature of providing an interface for rerandomizing previously generated proofs, all while preserving the zero-knowledge property.

2 Preliminaries

We give definitions of some primitives that help us to construct our protocols.

We let $(\mathcal{R}, z)$ denote the output of a relation generator $\mathfrak{R}(1^\lambda)$. $\mathcal{R}$ is a polynomial time decidable relation and z is an auxiliary input. For $(x;\omega) \in \mathcal{R}$, we call that x is the statement and ω is the witness. A non-interactive zero-knowledge system for $\mathcal{R}$ ($\mathsf{NIZK}_\mathcal{R}$) consists of the following algorithms:

- $\mathsf{NIZK}_\mathcal{R}.\mathsf{Setup}(1^\lambda) \to (\mathsf{crs}_\mathcal{R}, \mathsf{td}_\mathcal{R}, \mathsf{pp}_\mathcal{R})$: It outputs a common reference string $\mathsf{crs}_\mathcal{R}$, a trapdoor $\mathsf{td}_\mathcal{R}$ and a l public parameters $\mathsf{pp}_\mathcal{R}$ with respect to $\mathcal{R}$.
- $\mathsf{NIZK}_\mathcal{R}.\mathsf{Prove}(\mathsf{crs}_\mathcal{R}, \mathsf{pp}_\mathcal{R}, x; \omega) \to \pi$: It creates a proof π for $(x;\omega) \in \mathcal{R}$.
- $\mathsf{NIZK}_\mathcal{R}.\mathsf{Verify}(\mathsf{crs}_\mathcal{R}, \mathsf{pp}_\mathcal{R}, x; \pi)$ returns either 1 (verified) of 0 (not verified).
- $\mathsf{NIZK}_\mathcal{R}.\mathsf{Simulate}(\mathsf{td}_\mathcal{R}, \mathsf{pp}_\mathcal{R}, x) \to \pi$ returns a proof π.

NIZK satisfies the following:

Definition 1. *[Perfect Completeness] We say* $\mathsf{NIZK}_{\mathcal{R}}$ *has perfect completeness if* $\forall\lambda, \mathcal{R}$ *generated by* $\mathfrak{R}$ *and* $\forall(x;\omega) \in \mathcal{R}$*,* $\Pr[\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Verify}(\mathsf{crs}_{\mathcal{R}}, \mathsf{pp}_{\mathcal{R}}, x, \pi) \to 1 | \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Setup}(1^\lambda) \to (\mathsf{crs}_{\mathcal{R}}, \mathsf{td}_{\mathcal{R}}, \mathsf{pp}_{\mathcal{R}}), \pi \leftarrow \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Prove}(\mathsf{crs}_{\mathcal{R}}, \mathsf{pp}_{\mathcal{R}}, x; \omega)] = 1$.

Definition 2. *[Perfect Zero-Knowledge] We say* $\mathsf{NIZK}_{\mathcal{R}}$ *is* perfect zero-knowledge *if* $\forall\lambda, (\mathcal{R}, z)$ *generated by* $\mathfrak{R}$ *and* $\forall(x;\omega) \in \mathcal{R}$ *and all adversaries* $\mathcal{A}$ *the following holds given that* $(\mathsf{crs}_{\mathcal{R}}, \mathsf{td}_{\mathcal{R}}, \mathsf{pp}_{\mathcal{R}}) \leftarrow \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Setup}(1^\lambda)$*:*

$$\begin{aligned}&\Pr[\mathcal{A}(\mathsf{crs}_{\mathcal{R}}, \mathsf{pp}_{\mathcal{R}}, z, \pi, \mathcal{R}) = 1 \ |\pi \leftarrow \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Prove}(\mathsf{crs}_{\mathcal{R}}, \mathsf{pp}_{\mathcal{R}}, x; \omega)]\\ = &\Pr[\mathcal{A}(\mathsf{crs}_{\mathcal{R}}, \mathsf{pp}_{\mathcal{R}}, z, \pi, \mathcal{R}) = 1 \ |\pi \leftarrow \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Simulate}(\mathsf{td}_{\mathcal{R}}, \mathsf{pp}_{\mathcal{R}}, x)]\end{aligned}$$

Definition 3. *[Knowledge Soundness] We say* $\mathsf{NIZK}_{\mathcal{R}}$ *is knowledge sound if for any non-uniform PPT adversary* $\mathcal{A}$ *there exists a PPT extractor* $\mathcal{E}$ *such that*

$$\begin{aligned}&\Pr[\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Verify}(\mathsf{crs}_{\mathcal{R}}, \mathsf{pp}_{\mathcal{R}}, x, \pi) = 1 \wedge (x;\omega) \notin \mathcal{R} | (\mathcal{R}, z) \leftarrow \mathfrak{R},\\ &(\mathsf{crs}_{\mathcal{R}}, \mathsf{td}_{\mathcal{R}}, \mathsf{pp}_{\mathcal{R}}) \leftarrow \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Setup}(1^\lambda), ((x,\pi);\omega) \leftarrow (\mathcal{A}||\mathcal{E})(\mathcal{R}, z, \mathsf{crs}_{\mathcal{R}}, \mathsf{pp}_{\mathcal{R}})] = \mathsf{negl}(\lambda)\end{aligned}$$

where $(o_A; o_B) \leftarrow A||B(input)$ *denote the algorithms that run on the same input and* B *has access to the random coins of* A*.*

In our NIZK definition above the corresponding algorithms have more parameters that generally needed. This statement refers to the fact that not all $\mathsf{crs}_{\mathcal{R}}$, $\mathsf{pp}_{\mathcal{R}}$ or $\mathsf{td}_{\mathcal{R}}$ may be needed by a $\mathsf{NIZK}_{\mathcal{R}}$ algorithm for a given $\mathcal{R}$. In the rest of the work, we adhere to the convention that when instantiating the general $\mathsf{NIZK}_{\mathcal{R}}$ api for a specific $\mathcal{R}$, for simplicity, we will leave out the parameters which in that particular instantiation are the empty set. We do this in order to aggregate in one definition the different types of NIZK that we need and use in this work. Indeed, in case of the non-interactive version of a Sigma protocol, we have that $\mathsf{crs}_{\mathcal{R}} = \emptyset$ and $\mathsf{pp}_{\mathcal{R}} = \emptyset$. For a $\mathsf{NIZK}_{\mathcal{R}}$ such as Groth16 [29], $\mathsf{pp}_{\mathcal{R}} = \emptyset$. However, for our particular instantiation of $\mathsf{NIZK}_{\mathcal{R}}$ in Section 6 with $\mathcal{R}$ defined in Section 5, the $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Setup}$ outputs all three parameters $\mathsf{crs}_{\mathcal{R}}$, $\mathsf{td}_{\mathcal{R}}$, $\mathsf{pp}_{\mathcal{R}}$. Thus our definition allows for maximum flexibility. Finally, for each of our instantiations, we consider only benign auxiliary inputs as defined in [4].

Definition 4 (Non-interactive knowledge of arguments (NARK)). $\mathsf{NARK}_{\mathcal{R}}$ *for a relation* $\mathcal{R}$ *consists of the same algorithms in* NIZK *but satisfies only completeness (Definition 1) and knowledge soundness (Definition 3).*

Definition 5 (Commitment Scheme). Com *consists of the algorithms:*

- $\mathsf{Com}.\mathsf{Commit}(x) \mapsto c, r$ *outputs a commitment* c *to* x *and an opening* r*.*
- $\mathsf{Com}.\mathsf{Open}(c; x, r) \mapsto x'$ *opens the commitment* c *with the openings* x, r *to* x'*.*

If Com is a deterministic commitment scheme, we ignore r.

3 Security Model of Ring VRF

In this section, we define a ring VRF scheme in the UC framework, covering both real-world and ideal-world executions.

Definition 6 (Ring VRF). *It is defined with public parameters pp generated by a setup algorithm* $\mathsf{rVRF}.\mathsf{Setup}(1^\lambda)$ *and with the following PPT algorithms. All algorithms below include pp as part of their input, although it may not always be explicitly stated.*

- $\mathsf{rVRF.KeyGen}(pp) \rightarrow (\mathsf{sk}, \mathsf{pk})$*: It generates a secret key and public key pair* $(\mathsf{sk}, \mathsf{pk})$ *given input pp.*
- $\mathsf{rVRF.Eval}(\mathsf{sk}_i, \mathsf{in}) \rightarrow \mathsf{out}$*: It is a deterministic algorithm that outputs an evaluation value* $\mathsf{out} \in \mathcal{S}_{eval}$ *given* sk_i *and an input* in*. Here,* $\mathcal{S}_{eval} \in pp$ *and is the domain of evaluation values.*

The following algorithms need an input $\mathsf{ring} = \{\mathsf{pk}_1, \mathsf{pk}_2, \ldots, \mathsf{pk}_n\}$ *that we call ring:*

- $\mathsf{rVRF.CommitRing}(\mathsf{ring}, \mathsf{pk}_i) \rightarrow (\mathsf{comring}, \mathsf{opring})$*: It outputs a commitment of* ring *with the opening* opring *given input* ring *and* $\mathsf{pk} \in \mathsf{ring}$*.*
- $\mathsf{rVRF.OpenRing}(\mathsf{comring}, \mathsf{opring}) \rightarrow \mathsf{pk}$*: It outputs a public key* pk *given commitment* $\mathsf{comring}$ *and an opening* opring *of* $\mathsf{comring}$ *to* pk*.*
- $\mathsf{rVRF.Sign}(\mathsf{sk}_i, \mathsf{comring}, \mathsf{opring}, \mathsf{in}, \mathsf{ad}) \rightarrow \sigma$*: It outputs a signature* σ *of* $\mathsf{in}, \mathsf{ad} \in \{0,1\}^*$ *given* $\mathsf{sk}_i, \mathsf{opring}$ *and* $\mathsf{comring}$
- $\mathsf{rVRF.Verify}(\mathsf{comring}, \mathsf{in}, \mathsf{ad}, \sigma) \rightarrow (b, \mathsf{out})$*: It is a deterministic algorithm that outputs* $b \in \{0,1\}$ *and* $\mathsf{out} \in \mathcal{S}_{eval} \cup \{\perp\}$*.* $b = 1$ *means* σ *and* out *are verified.*

We note that $\mathsf{rVRF.CommitRing}$ and $\mathsf{rVRF.OpenRing}$ are optional algorithms of a ring VRF scheme. If they are not defined, we should let $\mathsf{comring} = \mathsf{ring}$ and $\mathsf{opring} = \mathsf{pk}$. $\mathsf{rVRF.CommitRing}$ and $\mathsf{rVRF.OpenRing}$ are useful for a succinct verification process in the case of a large ring.

We summary the security properties for rVRF informally as follows:

- *correctness*; when an honest signer with key $(\mathsf{sk}_i, \mathsf{pk}_i)$ outputs σ by running $\mathsf{rVRF.Sign}(\mathsf{sk}_i, \mathsf{comring}, \mathsf{opring}, \mathsf{in}, \mathsf{ad})$, $\mathsf{rVRF.Verify}(\mathsf{comring}, \mathsf{in}, \mathsf{ad}, \sigma)$ must output $1, \mathsf{out} = \mathsf{rVRF.Eval}(\mathsf{sk}_i, \mathsf{in})$ given $\mathsf{rVRF.OpenRing}(\mathsf{comring}, \mathsf{opring}) \rightarrow \mathsf{pk}_i \in \mathsf{ring}$. Indeed, while verifying the ring VRF signature, a verifier verifies that ad is signed by one of the keys is in the ring and also verifies that out is the evaluation value of in generated with the same key.
- *randomness*; out is random and independent from the input and the key.
- *anonymity* meaning that the output of $\mathsf{rVRF.Sign}$ does not leak any information about the key of its signer except that the key is in the ring.
- *unforgeability*; an adversary should not be able to forge a ring VRF signature
- *uniqueness*; the number of verified evaluation values should not be more than the number of the keys in the ring.

We remark that the output of $\mathsf{rVRF.Eval}$ is independent of any specific ring. Consequently, the verification of two signatures for a given input using different rings results in the same evaluation value. This property allows a party to disclose their identity as needed. For instance, suppose $\mathsf{out} \leftarrow \mathsf{rVRF.Eval}(\mathsf{sk}_i, \mathsf{in})$ is verified via a ring VRF signature σ with a ring containing pk_i. Later, if the corresponding party wishes to affirm that out was generated using their key, they simply need to sign the same input with a ring which consists of only their key i.e., $\mathsf{ring} = \{\mathsf{pk}_i\}$.

The ring VRF in the ideal world: We introduce a ring VRF functionality $\mathcal{F}_{\mathsf{rvrf}}$ to model execution of a ring VRF protocol in the ideal world. In other words, we define a ring VRF protocol in the case of having a trusted entity $\mathcal{F}_{\mathsf{rvrf}}$. There are many straightforward ways of defining a ring VRF protocol in the ideal world satisfying the desired security properties. However, defining simple and intuitive functionality while being as expressive and realizable in the real world execution is usually at odds [11]. Therefore, we have a lengthy $\mathcal{F}_{\mathsf{rvrf}}$ (See Figure 4) which satisfies the security properties that we expect from a ring VRF scheme and at the same time as faithful to the reality as possible. For the sake of clarity and accessibility, we split each execution part of $\mathcal{F}_{\mathsf{rvrf}}$ while we introduce our functionality. The composition of all parts is in Figure 4. We first describe how $\mathcal{F}_{\mathsf{rvrf}}$ works and then show which security properties it achieves.

$\mathcal{F}_{\mathsf{rvrf}}$ has tables to store the data generated from the requests from honest parties and the adversary Sim. The table `signing_keys` keeps the keys of parties. The other table `anonymous_key_map` stores an anonymous key that corresponds to an input of a party with a key pk. We note that the real execution of a ring VRF (Definition 6) does not have a concept of an anonymous key but $\mathcal{F}_{\mathsf{rvrf}}$ needs this internally to execute the verification of a ring signature. Related to anonymous keys, $\mathcal{F}_{\mathsf{rvrf}}$ also stores all malicious anonymous keys in a table $\mathcal{W}$. Finally, $\mathcal{F}_{\mathsf{rvrf}}$ stores the evaluations values of all parties in $\mathtt{evaluations}$. In a nutshell, given pk and in, $\mathcal{F}_{\mathsf{rvrf}}$ generates an anonymous key W as explained below and sets $\mathtt{anonymous\_key\_map}[\mathsf{in}, W]$ to pk. Then, it generates an evaluation value out as explained below and sets $\mathtt{evaluations}[\mathsf{in}, W]$ to out. In short, given honestly generated secret, public key pair $(\mathsf{sk}, \mathsf{pk})$ in the real world, the algorithm $\mathsf{rVRF.Eval}(\mathsf{sk}, \mathsf{in})$ that outputs evaluation value corresponds to generating an anonymous key W for pk, in and obtaining the evaluation value stored in $\mathtt{evaluations}[\mathsf{in}, W]$ in the ideal world. The necessity and usage of all these tables and anonymous keys will be more clear while we explain $\mathcal{F}_{\mathsf{rvrf}}$ in detail. $\mathcal{F}_{\mathsf{rvrf}}$ consists of the following execution parts.

Key Generation: When an honest party requests a key, $\mathcal{F}_{\mathsf{rvrf}}$ obtains a key pair $(\mathsf{sk}, \mathsf{pk})$ from Sim. $\mathcal{F}_{\mathsf{rvrf}}$ stores them if they have not been recorded. If it is the case, $\mathcal{F}_{\mathsf{rvrf}}$ gives only pk to the honest party. $\mathcal{F}_{\mathsf{rvrf}}$ will later use sk during signature generation. One can imagine sk as a secret key and pk as a public key but retrieving sk from Sim poses no issue in the ideal model. This is due to the fact that each evaluation value is randomly sampled, and a signature generated by an honest party can be considered valid if and only if they request it, as guaranteed by the verification process of $\mathcal{F}_{\mathsf{rvrf}}$.

[Key Generation.] upon receiving a message $(\mathsf{keygen}, \mathsf{sid})$ from P_i, send $(\mathsf{keygen}, \mathsf{sid}, \mathsf{P}_i)$ to the simulator Sim. Upon receiving a message $(\mathsf{verificationkey}, \mathsf{sid}, \mathsf{sk}, \mathsf{pk})$ from Sim, verify that sk or pk has not been recorded before for sid in $\mathtt{signing\_keys}$. If it is the case, store the value sk, pk in the table $\mathtt{signing\_keys}$ under P_i and return $(\mathsf{verificationkey}, \mathsf{sid}, \mathsf{pk})$ to P_i.

Honest Ring VRF Signature and Evaluation: This part of $\mathcal{F}_{\mathsf{rvrf}}$ functions for honest parties who evaluate an input $\in$ and sign a message ad and in. An honest party P_i provides to $\mathcal{F}_{\mathsf{rvrf}}$ a ring, its own public key pk_i, ad and in to be evaluated. Afterwards, $\mathcal{F}_{\mathsf{rvrf}}$ generates the evaluation value of in and pk_i and signs in and ad for a given ring if $\mathsf{pk}_i \in \mathsf{ring}$. The evaluation for honest parties works as follows: If $\mathcal{F}_{\mathsf{rvrf}}$ did not select any anonymous key for in and pk_i before, it samples randomly an anonymous key W and samples randomly the evaluation value out. The ring signature generation works as follows: $\mathcal{F}_{\mathsf{rvrf}}$ runs a PPT algorithm $\mathsf{Gen}_{sign}(\mathsf{ring}, \mathsf{sk}, \mathsf{pk}, \mathsf{ad}, \mathsf{in})$ where $(\mathsf{sk}, \mathsf{pk}) \in \mathtt{signing\_keys}$ and obtains a signature σ. It records $[\mathsf{in}, \mathsf{ad}, W, \mathsf{ring}, \sigma, 1]$ for verification. Here, 1 indicates that σ is a valid ring signature of in and ad generated for ring with the anonymous key W.

[Honest Ring VRF Signature and Evaluation.] upon receiving a message $(\mathsf{sign}, \mathsf{sid}, \mathsf{ring}, \mathsf{pk}_i, \mathsf{ad}, \mathsf{in})$ from P_i, verify that $\mathsf{pk}_i \in \mathsf{ring}$ and that there exists a public key pk_i associated to P_i in $\mathtt{signing\_keys}$. If it is not the case, just ignore the request. If there exists no W' such that $\mathtt{anonymous\_key\_map}[\mathsf{in}, W'] = \mathsf{pk}_i$, let $W \leftarrow\!\!\$\ \mathcal{S}_W$ and let $\mathsf{out} \leftarrow\!\!\$\ \mathcal{S}_{eval}$. Set $\mathtt{anonymous\_key\_map}[\mathsf{in}, W] = \mathsf{pk}_i$ and set $\mathtt{evaluations}[\mathsf{in}, W] = \mathsf{out}$. In any case (except ignoring), obtain W, out where $\mathtt{anonymous\_key\_map}[\mathsf{in}, W] = \mathsf{pk}_i$, $\mathtt{evaluations}[\mathsf{in}, W] = \mathsf{out}$ and $(\mathsf{sk}, \mathsf{pk})$ is in $\mathtt{signing\_keys}$. Then run $\mathsf{Gen}_{sign}(\mathsf{ring}, \mathsf{sk}, \mathsf{pk}, \mathsf{ad}, \mathsf{in}) \to \sigma$. Let $\sigma = (\sigma, W)$ and record $[\mathsf{in}, \mathsf{ad}, W, \mathsf{ring}, \sigma, 1]$. Return $(\mathsf{signature}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, \mathsf{in}, \mathsf{out}, \sigma)$ to P_i.

Malicious Ring VRF Evaluation: This part is designed for Sim to evaluate an input in with an anonymous key. For this, it provides to $\mathcal{F}_{\mathsf{rvrf}}$ in, a malicious key pk and an anonymous key W. Then, $\mathcal{F}_{\mathsf{rvrf}}$ evaluates in with pk if an anonymous key $W' \neq W$ is not assigned to in and pk before. If it is the case, it returns the randomly selected evaluation value stored in $\mathtt{evaluations}[\mathsf{in}, W]$. The reason of conditioning on a unique anonymous key for in and pk is to prevent Sim to obtain more than one evaluation values for in and pk. This is necessary for the uniqueness property. We remark that it is possible for Sim to obtain the same evaluation value of in with two different malicious keys $\mathsf{pk}_i, \mathsf{pk}_j$ by sending $(\mathsf{eval}, \mathsf{sid}, \mathsf{pk}_i, W, \mathsf{in})$ and $(\mathsf{eval}, \mathsf{sid}, \mathsf{pk}_j, W, \mathsf{in})$. However, this does not break the uniqueness.

[Malicious Ring VRF Evaluation.] upon receiving a message $(\mathsf{eval}, \mathsf{sid}, \mathsf{pk}_i, W, \mathsf{in})$ from Sim, if pk_i is recorded under an honest party's identity or if there exists $W' \neq W$ where $\mathtt{anonymous\_key\_map}[\mathsf{in}, W'] = \mathsf{pk}_i$, ignore the request. Otherwise, record in the table $\mathtt{signing\_keys}$ the value $(\perp, \mathsf{pk}_i)$ under Sim if $(., \mathsf{pk}_i)$ is not in $\mathtt{signing\_keys}$. If $\mathtt{anonymous\_key\_map}[\mathsf{in}, W]$ is not defined before, set $\mathtt{anonymous\_key\_map}[\mathsf{in}, W] = \mathsf{pk}_i$ and let $\mathsf{out} \leftarrow\!\!\$ \ \mathcal{S}_{eval}$ and set $\mathtt{evaluations}[\mathsf{in}, W] = \mathsf{out}$. In any case (except ignoring), obtain $\mathsf{out} = \mathtt{evaluations}[\mathsf{in}, W]$ and return $(\mathsf{evaluated}, \mathsf{sid}, \mathsf{in}, \mathsf{pk}_i, W, \mathsf{out})$ to P_i.

We remark that if Sim provides an anonymous key W of any honest party during the evaluation process, Sim can learn the evaluation of in for this honest party without needing to know who is this party. For this, it just needs to send the message $(\mathsf{eval}, \mathsf{sid}, \mathsf{pk}_i, W, \mathsf{in})$ where pk_i is any verification key. In such a case, $\mathcal{F}_{\mathsf{rvrf}}$ returns immediately $\mathtt{evaluations}[\mathsf{in}, W]$ without checking whether $\mathtt{anonymous\_key\_map}[\mathsf{in}, W] = \mathsf{pk}_i$. So if $\mathtt{anonymous\_key\_map}[\mathsf{in}, W]$ belongs to an honest party, Sim learns the evaluation value of some honest party but does not who they are. We note that this leakage does not contradict the desired security properties and helps us to prove our ring VRF protocols realizes $\mathcal{F}_{\mathsf{rvrf}}$.

Requests of Signatures: If Sim provides $W, \mathsf{ad}, \mathsf{in}$, Sim obtains all valid and stored ring signatures of in and ad generated with an anonymous key W.

[Malicious Requests of Signatures.] upon receiving a message $(\mathsf{signs}, \mathsf{sid}, W, \mathsf{ad}, \mathsf{in})$ from Sim, obtain all existing valid signatures σ such that $[\mathsf{in}, \mathsf{ad}, W, ., \sigma, 1]$ is recorded and add them in a list $\mathcal{L}_\sigma$. Return $(\mathsf{signs}, \mathsf{sid}, W, \mathsf{ad}, \mathsf{in}, \mathcal{L}_\sigma)$ to Sim.

Ring VRF Verification: This part of $\mathcal{F}_{\mathsf{rvrf}}$ is to check whether σ signs in and ad for ring with anonymous key W. This part corresponds to $\mathsf{rVRF.Verify}$ in the real world ring VRF protocol. Therefore, $\mathcal{F}_{\mathsf{rvrf}}$ first checks various conditions to decide if the signature is valid. If the signature is verified, $\mathcal{F}_{\mathsf{rvrf}}$ outputs $b = 1$ and $\mathsf{out} = \mathtt{evaluations}[\mathsf{in}, W]$. Otherwise, it outputs $b = 0$ and $\mathsf{out} = \perp$.

For the verification of the signature, $\mathcal{F}_{\mathsf{rvrf}}$ first checks its records to see whether this signature is verified or unverified in its records i.e., checks whether $[\mathsf{in}, \mathsf{ad}, W, \mathsf{ring}, \sigma, b']$ is recorded (See C1). If it is recorded, $\mathcal{F}_{\mathsf{rvrf}}$ lets $b = b'$ to be consistent. Otherwise, it checks whether W is an anonymous key of an honest party generated for in (See C2). If it is the case, $\mathcal{F}_{\mathsf{rvrf}}$ checks its records whether this honest party requested signing in and ad for ring. If there exists such record i.e., $[\mathsf{in}, \mathsf{ad}, W, \mathsf{ring}, ., 1]$, it stores the new signature σ as a valid signature in its records and lets $b = 1$. We remark that Sim can create arbitrary verified signatures that sign any in and ad for ring with W once the honest party owning W has requested signing in and ad for ring. This does not break the forgeability property because the honest party has already signed for it. If none of the above conditions (C1 and C2) holds, it means that σ could be a signature generated for a malicious party. Therefore, $\mathcal{F}_{\mathsf{rvrf}}$ asks about it to Sim and Sim replies with a public key $\mathsf{pk}_{\mathsf{Sim}}$ and an indicator b_{Sim} showing that σ is valid or invalid. Then, $\mathcal{F}_{\mathsf{rvrf}}$ checks various conditions to prevent Sim forging and violating the uniqueness. To prevent forging, it lets directly $b = 0$, if $\mathsf{pk}_{\mathsf{Sim}}$ is a key of an honest party. If $\mathsf{pk}_{\mathsf{Sim}}$ is not an honest key, then $\mathcal{F}_{\mathsf{rvrf}}$ checks its table $\mathcal{W}[\mathsf{in}, \mathsf{ring}]$ which stores the anonymous keys of valid malicious signatures of in for ring. If the number of anonymous keys in $\mathcal{W}[\mathsf{in}, \mathsf{ring}]$ is greater than or equal to the number of malicious keys in

ring, then $\mathcal{F}_{\mathsf{rvrf}}$ invalidates σ by letting $b = 0$. This condition guarantees uniqueness meaning that the number of verifying evaluation values that Sim can generate for in with ring is at most the number of malicious keys in ring. If the number of malicious anonymous keys of valid signatures does not exceed the number of malicious keys in ring, then $\mathcal{F}_{\mathsf{rvrf}}$ checks whether W is a unique anonymous key assigned to $\mathsf{in}, \mathsf{pk}_{\mathsf{Sim}}$ as in the "Malicious Ring VRF Evaluation". If W is unique then $\mathcal{F}_{\mathsf{rvrf}}$ lets $b = b_{\mathsf{Sim}}$.

After deciding b, $\mathcal{F}_{\mathsf{rvrf}}$ records it as $[\mathsf{in}, \mathsf{ad}, W, \mathsf{ring}, \sigma, b]$ to be able to reply with the same b for the same verification query later. If $b = 1$, $\mathcal{F}_{\mathsf{rvrf}}$ returns $\mathtt{evaluations}[\mathsf{in}, W]$ as well.

[Ring VRF Verification.] upon receiving a message $(\mathsf{verify}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, \mathsf{in}, \sigma)$ from a party, do the following:

C1 If there exits a record $[\mathsf{in}, \mathsf{ad}, W, \mathsf{ring}, \sigma, b']$, set $b = b'$.
C2 Else if $\mathtt{anonymous\_key\_map}[\mathsf{in}, W]$ is an honest verification key and there exists a record $[\mathsf{in}, \mathsf{ad}, W, \mathsf{ring}, \sigma', 1]$ for any σ', then let $b = 1$ and record $[\mathsf{in}, \mathsf{ad}, W, \mathsf{ring}, \sigma, 1]$.
C3 Else relay the message $(\mathsf{verify}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, \mathsf{in}, \sigma)$ to Sim and receive back the message $(\mathsf{verified}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, \mathsf{in}, \sigma, b_{\mathsf{Sim}}, \mathsf{pk}_{\mathsf{Sim}})$. Then check the following:
1. If $\mathsf{pk}_{\mathsf{Sim}}$ is an honest verification key, set $b = 0$.
2. Else if $W \notin \mathcal{W}[\mathsf{in}, \mathsf{ring}]$ and $|\mathcal{W}[\mathsf{in}, \mathsf{ring}]| \geq |\mathsf{ring}_{mal}|$ where ring_{mal} is a set of malicious keys in ring, set $b = 0$. .
3. Else if there exists $W' \neq W$ where $\mathtt{anonymous\_key\_map}[\mathsf{in}, W'] = \mathsf{pk}_{\mathsf{Sim}}$, set $b = 0$.
4. Else set $b = b_{\mathsf{Sim}}$.

In the end, record $[\mathsf{in}, \mathsf{ad}, W, \mathsf{ring}, \sigma, 0]$ if it is not stored. If $b = 0$, let $\mathsf{out} = \perp$. Otherwise, do the following:

- if $W \notin \mathcal{W}[\mathsf{in}, \mathsf{ring}]$, add W to $\mathcal{W}[\mathsf{in}, \mathsf{ring}]$.
- if $\mathtt{evaluations}[\mathsf{in}, W]$ is not defined, sample $y \leftarrow\!\!\$\ \mathcal{S}_{eval}$. Then, set $\mathtt{anonymous\_key\_map}[\mathsf{in}, W] = \mathsf{pk}_{\mathsf{Sim}}$ and $\mathtt{evaluations}[\mathsf{in}, W] = \mathsf{out}$.
- otherwise, set $\mathsf{out} = \mathtt{evaluations}[\mathsf{in}, W]$.

Finally, output $(\mathsf{verified}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, \mathsf{in}, \sigma, \mathsf{out}, b)$ to the party.

In the real-world ring VRF, the verification algorithm outputs the corresponding evaluation value of the signer. Therefore, $\mathcal{F}_{\mathsf{rvrf}}$ outputs the signer's evaluation value if the signature is verified. However, it achieves this together with the anonymous key which is not defined in the ring VRF in the real world. If $\mathcal{F}_{\mathsf{rvrf}}$ did not define an anonymous key for each signature, then there would be no way that $\mathcal{F}_{\mathsf{rvrf}}$ determines the signer's key and outputs the evaluation value because σ does not need to be unique for each key. Therefore, $\mathcal{F}_{\mathsf{rvrf}}$ maps a random and independent anonymous key to each in and pk so that this key behaves as if it is the verification key of the signature. Since it is random and independent from in and pk, it does not leak any information about the party during the verification but it still allows $\mathcal{F}_{\mathsf{rvrf}}$ to distinguish the signer.

We remark that when $\mathcal{F}_{\mathsf{rvrf}}$ is in C3, it does not check whether the provided public key $\mathsf{pk}_{\mathsf{Sim}}$ is in the ring. This allows Sim to generate a signature of in for ring that is signed by $\mathsf{pk}_{\mathsf{Sim}}$, even if $\mathsf{pk}_{\mathsf{Sim}}$ is not necessarily a part of ring. However, it does not break any security properties that we aim for a ring VRF scheme as it can be seen in the analysis of $\mathcal{F}_{\mathsf{rvrf}}$ below.

Corruption: Sim can corrupt any honest party at any time. So, $\mathcal{F}_{\mathsf{rvrf}}$ provides security against an adaptive adversary.

[Corruption:] upon receiving $(\mathsf{corrupt}, \mathsf{sid}, \mathsf{P}_i)$ from Sim, remove (x_i, pk_i) from $\mathtt{signing\_keys}[\mathsf{P}_i]$ and store them to $\mathtt{signing\_keys}$ under Sim. Return $(\mathsf{corrupted}, \mathsf{sid}, \mathsf{P}_i)$.

This is the end of description $\mathcal{F}_{\mathsf{rvrf}}$. It is not immediately evident which security properties our functionality provides. Therefore, we now proceed to analyse these properties. Throughout our analysis, the evaluation value of $(\mathsf{in}, \mathsf{pk}_i)$ refers to $\mathtt{evaluations}[\mathsf{in}, W]$ where $\mathtt{anonymous\_key\_map}[\mathsf{in}, W] = \mathsf{pk}_i$.

Randomness: $\mathcal{F}_{\mathsf{rvrf}}$ satisfies the following randomness property: The evaluation value of $(\mathsf{in}, \mathsf{pk}_i)$ is independently and randomly selected for all honest keys pk_i. Likewise, the evaluation value of pairs $(\mathsf{in}, \mathsf{pk}_i)$ with an anonymous key W provided by Sim is also randomly selected independently for all malicious keys pk_i. We remark that since Sim can provide the same anonymous key for different public keys for the same input in, we consider the randomness of an evaluation value that is generated for all pairs $\{(\mathsf{in}, \mathsf{pk}_i)\}$ sharing the same anonymous key in the case of malicious evaluations.

Determinism: $\mathcal{F}_{\mathsf{rvrf}}$ satisfies that the evaluation value of $(\mathsf{in}, \mathsf{pk}_i)$, once it has been evaluated, is unique and cannot be changed.

The reason of it is that once an anonymous key W is assigned to $(\mathsf{in}, \mathsf{pk}_i)$, it cannot be updated. Therefore, when this happen, $\mathtt{evaluations}[\mathsf{in}, W]$ is fixed leading to output always the same evaluation value.

Unforgeability: If an honest party with a public key pk never signs an input in and an associated data ad for a ring, then no other party can generate a forgery of in and ad for ring signed by pk. Formally, if an honest party with pk never sends a message $(\mathsf{sign}, \mathsf{sid}, \mathsf{ring}, \mathsf{pk}, \mathsf{ad}, \mathsf{in})$ for some $\mathsf{ring}, \mathsf{in}, \mathsf{ad}$, then no party can create a record $[\mathsf{in}, \mathsf{ad}, W, \mathsf{ring}, ., 1]$ in $\mathcal{F}_{\mathsf{rvrf}}$ where $\mathtt{anonymous\_key\_map}[\mathsf{in}, \mathsf{pk}] = W$.

To analyse this, we need to check the places where $\mathcal{F}_{\mathsf{rvrf}}$ records a valid signature for an honest party. The first place is during the process of honest ring VRF signature and evaluation. Here, $\mathcal{F}_{\mathsf{rvrf}}$ records a valid signature if an honest party having a key pk sends a message $(\mathsf{sign}, \mathsf{sid}, \mathsf{ring}, \mathsf{pk}, \mathsf{ad}, \mathsf{in})$ to $\mathcal{F}_{\mathsf{rvrf}}$. Therefore, Sim cannot create a forgery there. The other place is during the verification process. $\mathcal{F}_{\mathsf{rvrf}}$ creates a valid signature record in C2 if the corresponding honest party has already signed for in, ad for ring. So, forgery is not possible in C2 as well. It also creates a valid signature record in C3. However, $\mathcal{F}_{\mathsf{rvrf}}$ never records a valid signature for an honest party here because it forbids it by C3-1.

Uniqueness: An evaluation value out for an input in is verified with ring, if there exists a signature σ such that $\mathcal{F}_{\mathsf{rvrf}}$ returns $(\mathsf{out}, 1)$ for a query $(\mathsf{verify}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, \mathsf{in}, \sigma)$ for some anonymous key W and message ad. The uniqueness property guarantees that the number of verified evaluation values of an input in with ring is not more than $|\mathsf{ring}|$. $\mathcal{F}_{\mathsf{rvrf}}$ satisfies uniqueness:

If $\mathcal{F}_{\mathsf{rvrf}}$ outputs $(1, \mathsf{out})$ for a query $(\mathsf{verify}, \mathsf{sid}, \mathsf{ring}, W, ., \mathsf{in}, \sigma)$, it means that there exists a record $[\mathsf{in}, ., W, \mathsf{ring}, \sigma, 1]$ and $\mathsf{out} = \mathtt{evaluations}[\mathsf{in}, W]$, $\mathtt{anonymous\_key\_map}[\mathsf{in}, W] = \mathsf{pk}$. If pk is an honest key, then it means that $\mathsf{pk} \in \mathsf{ring}$ because $\mathcal{F}_{\mathsf{rvrf}}$ generates a signature for an honest party with a key if $\mathsf{pk} \in \mathsf{ring}$. Now, let's assume $\mathcal{F}_{\mathsf{rvrf}}$ does not satisfy uniqueness i.e., there exist t different verified evaluation values $\mathcal{O} = \{\mathsf{out}_1, \mathsf{out}_2, \ldots, \mathsf{out}_t\}$ of an input in with ring where $|\mathsf{ring}| = t - 1$. This implies that for each $\mathsf{out}_i \in \mathcal{O}$, there exists a record $[\mathsf{in}, ., W_i, \mathsf{ring}, \sigma_i, 1]$ such that $\mathtt{evaluations}[\mathsf{in}, W_i] = \mathsf{out}_i$ where $\mathtt{anonymous\_key\_map}[\mathsf{in}, W_i] = \mathsf{pk}_i$ and $W_i \neq W_j$ for all $i, j \in [1, t]$. Since $\mathcal{F}_{\mathsf{rvrf}}$ makes sure that there cannot be two different anonymous keys mapping to same $(\mathsf{in}, \mathsf{pk})$, $\mathsf{pk}_i \neq \mathsf{pk}_j$ for all $i \neq j \in [1, t]$. If pk_i is an honest key, it means that σ_i is not a forgery so $\mathsf{pk}_i \in \mathsf{ring}$. Therefore, each honest evaluation value in $\mathcal{O}$ maps to one honest public key in ring meaning that honest evaluation values in $\mathcal{O}$ is at most $|\mathsf{ring} \setminus \mathsf{ring}_{mal}| = n_h$. If pk_i is not an honest key, $W_i \in \mathcal{W}[\mathsf{in}, \mathsf{ring}]$ since $\mathcal{F}_{\mathsf{rvrf}}$ adds W_i to $\mathcal{W}[\mathsf{in}, \mathsf{ring}]$ whenever it creates such record for a malicious signature. $\mathcal{F}_{\mathsf{rvrf}}$ makes sure that in the condition C3-2 that $\mathcal{W}[\mathsf{in}, \mathsf{ring}] \leq |\mathsf{ring}_{mal}| = n_m$. Therefore, $t \leq n_h + n_m = |\mathsf{ring}|$ which is a contradiction.

Robustness: Sim cannot prevent an honest party to evaluate, sign or verify. The only place that $\mathcal{F}_{\mathsf{rvrf}}$ does not respond any query is when it aborts. It happens when it selects an honest anonymous key which already existed. This happens in negligible probability in λ.

Anonymity: We expect from an anonymous $\mathcal{F}_{\mathsf{rvrf}}$ to adhere to the condition that an honest signature σ generated for an input in with Gen_{sign} along with its associated anonymous key W should not give any information regarding the honest party's key, except for the fact that it is a member of ring. However, this condition should hold unless in has been signed by the same party for any other ring. In such a case, since both signatures includes W, the anonymity may be compromised i.e., Sim learns the party's key is in the intersection of ring and ring'. We note that this design choice is intentional, as it provides parties with the flexibility to reveal their identity when necessary.

It is evident that anonymous keys do not give any information related to honest party's key as they are randomly sampled by $\mathcal{F}_{\mathsf{rvrf}}$. However, this cannot be conclusively asserted for the signatures, because it depends on the specification of Gen_{sign}. Therefore, we introduce an anonymity definition (See Definition 7) for Gen_{sign} and establish that $\mathcal{F}_{\mathsf{rvrf}}$ is anonymous if Gen_{sign} is anonymous according to this definition.

Definition 7 (Anonymity of Gen_{sign}). *We define an anonymity game between a PPT distinguisher $\mathcal{D}$ and a challenger. In the game, $\mathcal{D}$ sends the query $(\mathsf{challenge}, \mathsf{ring}, (\mathsf{sk}_0, \mathsf{pk}_0), (\mathsf{sk}_1, \mathsf{pk}_1), \mathsf{in}, \mathsf{ad})$. Then the challenger checks if $\mathsf{pk}_0, \mathsf{pk}_1 \in \mathsf{ring}$. If it is the case, the challenger samples randomly $b \in \{0,1\}$ and runs $\mathsf{Gen}_{sign}(\mathsf{ring}, \mathsf{sk}_b, \mathsf{pk}_b, \mathsf{ad}, \mathsf{in}) \to \sigma_b$. It gives σ_b as a challenge to $\mathcal{D}$. In the end of the game, if $\mathcal{D}$ outputs $b' = b$, then wins the game.*

We say that Gen_{sign} is anonymous if any PPT distinguisher $\mathcal{D}$ has a negligible advantage in λ to win the anonymity game.

4 The First Ring VRF Construction

$\mathsf{rVRF.Setup}(1^\lambda)$ generates the public parameters $\mathsf{pp}_{rvrf} = (crs_{\mathsf{ring}}, crs_{\mathsf{comring}}, p, \mathbb{G}, G, K, \mathcal{S}_{eval} = \mathbb{F}_p)$. Here, p is a prime order of the group $\mathbb{G}$ with generators G, K. $crs_{\mathsf{ring}}, crs_{\mathsf{comring}}$ are generated by $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{ring}}}.\mathsf{Setup}(1^\lambda)$ and $\mathsf{NARK}_{\mathcal{R}_{\mathsf{comring}}}.\mathsf{Setup}(1^\lambda)$, respectively. Our ring VRF construction deploys random oracles $H_p, H : \{0,1\}^* \to \mathbb{F}_p$, $H_{\mathbb{G}} : \{0,1\}^* \to \mathbb{G}$.

We build our ring VRF protocol with an efficient evaluation proof, which we call the Pedersen VRF denoted PedVRF.

Pedersen VRF: We construct PedVRF following a similar approach as other VRF constructions [40,41,28]. The distinctions in PedVRF are that it does not expose any public key and the public key in these constructions used for verification is replaced by a Pedersen commitment to the secret key sk.

- $\mathsf{PedVRF.KeyGen}$ outputs $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{F}_p$.
- $\mathsf{PedVRF.Eval}(\mathsf{sk}, \mathsf{in}) \to \mathsf{out}$: It outputs the evaluation value of in which is $\mathsf{out} = H(\mathsf{in}, \mathsf{preout})$ where $\mathsf{preout} = \mathsf{sk}\, H_{\mathbb{G}}(\mathsf{in})$.
- $\mathsf{PedVRF.CommitKey}(\mathsf{sk}) \to (\mathsf{compk}, \mathsf{b})$: It selects randomly a blinding factor $\mathsf{b} \in \mathbb{F}_p$ and outputs the Pedersen commitment $\mathsf{compk} = \mathsf{sk}\, G + \mathsf{b}\, K$ and b.

Sign and Verify algorithms of PedVRF are directly aligned with the proving system of Chaum-Pedersen DLEQ for relation $\mathcal{R}_{eval}$ (see below), instantiated by a Fiat-Shamir transform of a sigma protocol.

$$\mathcal{R}_{eval} = \left\{ \begin{array}{l|l} (\mathsf{compk}, \mathsf{preout}, \mathsf{in}); & \mathsf{compk} = \mathsf{sk}\, G + \mathsf{b}\, K, \\ (\mathsf{sk}, \mathsf{b}) & \mathsf{preout} = \mathsf{sk}\, H_{\mathbb{G}}(\mathsf{in}) \end{array} \right\}.$$

– $\mathsf{PedVRF.Sign}(\mathsf{sk}, \mathsf{b}, \mathsf{in}, \mathsf{ad}) \to \sigma$: It receives as an input a secret key sk, a blinding factor $b \in \mathbb{F}_p$, an input in to prove its evaluation and an associated data ad to sign. It first computes $\mathsf{preout} := \mathsf{sk}\, H_{\mathbb{G}}(\mathsf{in})$ and $\mathsf{compk} = \mathsf{sk}G + \mathsf{b}K$. Then, it runs $\mathsf{NIZK}_{\mathcal{R}_{eval}}.\mathsf{Prove}(\mathsf{compk}, \mathsf{preout}, \mathsf{in}; \mathsf{sk}, \mathsf{b})$ which generates a Chaum-Pedersen DLEQ proof for relation $\mathcal{R}_{eval}$ i.e., let $r_1, r_2 \leftarrow\!\$\, \mathbb{F}_p$ and compute $R = r_1 G + r_2 K, R_m = r_1 H_{\mathbb{G}}(\mathsf{in})$ and $c = H_p(\mathsf{ad}, \mathsf{in}, \mathsf{compk}, \mathsf{preout}, R, R_m)$, finally compute $s_1 = r_1 + c\,\mathsf{sk}$ and $s_2 = r_2 + c\,\mathsf{b}$ and let $\pi = (c, s_1, s_2)$. In the end, it returns the signature $\sigma = (\pi, \mathsf{preout})$.
– $\mathsf{PedVRF.Verify}(\mathsf{compk}, \mathsf{in}, \mathsf{ad}, \sigma) \to (\mathsf{out} \vee \bot)$: It verifies σ with compk by running $\mathsf{NIZK}_{\mathcal{R}_{eval}}.\mathsf{Verify}(\mathsf{compk}, \mathsf{preout}, \mathsf{in}; \pi_{eval})$ i.e., parse $\sigma = (\mathsf{preout}, c, s_1, s_2)$ and check if $c = H_p(\mathsf{ad}, \mathsf{in}, \mathsf{compk}, \mathsf{preout}, R, R_m)$ where $R = s_1 G + s_2 K - c\,\mathsf{compk}$ and $R_m = s_1 H_{\mathbb{G}}(\mathsf{in}) - c\,\mathsf{preout}$. If this verifies, it outputs $H(\mathsf{in}, \mathsf{preout})$. Otherwise, it outputs failure $\bot$.

The verifier in PedVRF verifies that the secret key computed to generate preout and the secret key used to generate compk are the same. Therefore, $H(\mathsf{in}, \mathsf{preout})$ is the correct evaluation value of in with this secret key since preout is correct. In addition to this, they verify that ad is signed by the same key since π functions akin to a Schnorr-like signature. We note that PedVRF is not a VRF due to the absence of a public key but it can be transformed into EC-VRF [40,41,28] if the conditions $\mathsf{b} = r_2 = 0$ in Sign and $\mathsf{pk} = \mathsf{sk}G$ are imposed.

Now, we are ready to describe our first ring VRF construction.

The Ring VRF Construction: The main building blocks of our construction are PedVRF, $\mathsf{NIZK}_{\mathcal{R}_{\mathtt{ring}}}$, $\mathsf{NARK}_{\mathcal{R}_{\mathsf{comring}}}$ (relations defined below) and two commitment schemes Com and Com^* which is a deterministic commitment scheme.

– $\mathsf{rVRF.KeyGen}(\mathsf{pp}_{rvrf}) \to (\mathsf{sk}, \mathsf{pk})$: It outputs as secret key $\mathsf{sk} = (x, r)$ where $x, r \leftarrow\!\$\, \mathbb{F}_p$ and pk as public key where $\mathsf{pk} = \mathsf{Com.Commit}(x, r)$. We deploy this key generation algorithm based on a commitment scheme to be consistent with the key generation algorithm of our second construction in §6 which necessitates commitment to sk to run securely. However, we note that an alternative definition for pk is possible, where $\mathsf{pk} = \mathsf{sk}G$ and $\mathsf{sk} = x$.
– $\mathsf{rVRF.Eval}(\mathsf{sk}, \mathsf{in})$ runs $\mathsf{PedVRF.Eval}(x, \mathsf{in})$. We remark that the evaluation value is generated with *only* the first part of the secret key which is x.
– $\mathsf{rVRF.CommitRing}(\mathsf{ring}, \mathsf{pk}) \to (\mathsf{comring}, \mathsf{opring})$: It runs $\mathsf{Com}^*.\mathsf{Commit}(\mathsf{ring})$ and obtains $\mathsf{comring}$ as a deterministic commitment to ring. Then, it runs $\mathsf{NARK}_{\mathcal{R}_{\mathsf{comring}}}.\mathsf{Prove}(\mathsf{comring}, \mathsf{pk}; \mathsf{ring})$ which outputs π_{comring}. In the end, it outputs $\mathsf{opring} = (\mathsf{pk}, \pi_{\mathsf{comring}})$.

$$\mathcal{R}_{\mathsf{comring}} = \{(\mathsf{comring}, \mathsf{pk}; \mathsf{ring}) : \mathsf{Com}^*.\mathsf{Commit}(\mathsf{ring}) \to \mathsf{comring} \wedge \mathsf{pk} \in \mathsf{ring}\}$$

– $\mathsf{rVRF.OpenRing}(\mathsf{comring}, \mathsf{opring}) \to (\mathsf{pk} \vee \bot)$: It runs $\mathsf{NARK}_{\mathcal{R}_{\mathsf{comring}}}.\mathsf{Verify}(\mathsf{comring}, \mathsf{opring})$ where $\mathsf{opring} = (\mathsf{pk}, \pi_{\mathsf{comring}})$. If it verifies it outputs pk. Otherwise, it outputs $\bot$.
Here, we deploy NARK to show that committed ring contains a given public key. This enables us to instantiate the signing and verification algorithms of our protocols without requiring full knowledge of the ring. It is particularly crucial for applications that involve large-scale rings with millions of users.

The Sign and Verify for our rVRF are a combination of Sign and Verify from PedVRF and Prove and Verify from $\mathsf{NIZK}_{\mathcal{R}_{\mathtt{ring}}}$, as follows:

– $\mathsf{rVRF.Sign}(\mathsf{sk}, \mathsf{comring}, \mathsf{opring}, \mathsf{in}, \mathsf{ad}) \to \sigma$: It returns a ring VRF signature $\sigma = (\mathsf{compk}, \pi_{\mathtt{ring}}, \mathsf{comring}, \sigma')$. For this, it obtains $(\mathsf{b}, \mathsf{compk})$ by running $\mathsf{PedVRF.CommitKey}(\mathsf{sk})$ and runs $\mathsf{NIZK}_{\mathcal{R}_{\mathtt{ring}}}.\mathsf{Prove}(\mathsf{compk}, \mathsf{comring}; \mathsf{b}, \mathsf{opring}, \mathsf{pk}, \mathsf{sk}) \to \pi_{\mathtt{ring}}$, then obtains the Pedersen VRF signature σ' by running $\mathsf{PedVRF.Sign}(x, \mathsf{b}, \mathsf{in}, \mathsf{ad}')$ where $\mathsf{ad}' \leftarrow \mathsf{ad} \mathbin{+\!\!+} \pi_{\mathtt{ring}} \mathbin{+\!\!+} \mathsf{comring}$. Here,

$$\mathcal{R}_{\mathtt{ring}} = \left\{ (\mathsf{compk}, \mathsf{comring}; \mathsf{b}, \mathsf{opring}, \mathsf{sk} = (x, r)) \;\middle|\; \begin{array}{l} \mathsf{pk} = \mathsf{OpenRing}(\mathsf{comring}, \mathsf{opring}), \\ x = \mathsf{Com.Open}(\mathsf{pk}, r), \\ \mathsf{compk} = xG + \mathsf{b}K \end{array} \right\}$$

We note that if $\mathsf{pk} = \mathsf{sk}G$ then $\mathcal{R}_{\mathtt{ring}}$ does not need sk as a part of its witness. In this case, we need to replace the last two conditions by $\mathsf{compk} = \mathsf{pk} + \mathsf{b}K$.

– $\mathsf{rVRF.Verify}(\mathsf{comring}, \mathsf{in}, \mathsf{ad}, \sigma) \to (1, \mathsf{out}) \vee (0, \perp)$: It parses σ as $(\mathsf{compk}, \pi_{\mathtt{ring}}, \mathsf{comring}, \sigma')$, sets $\mathsf{ad}' \leftarrow \mathsf{ad} \mathbin{+\!\!+} \pi_{\mathtt{ring}} \mathbin{+\!\!+} \mathsf{comring}$ and runs $\mathsf{NIZK}_{\mathcal{R}_{\mathtt{ring}}}.\mathsf{Verify}((\mathsf{compk}, \mathsf{comring}); \pi_{\mathtt{ring}})$. If it fails, returns $(0, \perp)$. Otherwise, returns $\mathsf{PedVRF.Verify}(\mathsf{compk}, \mathsf{in}, \mathsf{ad}', \sigma)$.

We prove in Theorem 1 that our first ring VRF construction realizes $\mathcal{F}_{\mathsf{rvrf}}$ in Figure 4 but we want to give an intuition first why our scheme is secure. Intuitively, the randomness and the determinism of $\mathsf{rVRF.Eval}$ come from the random oracles H and $H_{\mathbb{G}}$. The anonymity of our ring VRF signature $(\sigma = (\mathsf{compk}, \pi_{ring}, \mathsf{preout}, \pi_{eval}, \mathsf{comring}))$ comes from the perfect hiding property of Pedersen commitment i.e., compk is independent from the signer's key, the zero-knowledge property of $\mathsf{NIZK}_{\mathcal{R}_{ring}}$ and $\mathsf{NIZK}_{\mathcal{R}_{eval}}$ and the difficulty of DDH in $\mathbb{G}$ (Lemma 2) so that preout is indistinguishable from a random element in $\mathbb{G}$. The unforgeability and uniqueness come from the fact that CDH is hard in $\mathbb{G}$ (Lemma 3).

Algorithm 1 $\mathsf{Gen}_{sign}(\mathsf{ring}, \mathsf{sk} = (x, r), \mathsf{pk}, \mathsf{ad}, \mathsf{in})$

```
1: $c, s_1, s_2 \leftarrow\!\$\, \mathbb{F}_p$
2: $\pi_{eval} \leftarrow (c, s_1, s_2)$
3: $\mathsf{b} \leftarrow\!\$\, \mathbb{F}_p$
4: $\mathsf{compk} = xG + \mathsf{b}\,K$
5: $\mathsf{comring}, \mathsf{opring} \leftarrow \mathsf{rVRF.CommitRing}(\mathsf{ring}, \mathsf{pk})$
6: $\pi_{ring} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{\mathtt{ring}}}.\mathsf{Prove}(\mathsf{crs}_{\mathcal{R}_{\mathtt{ring}}}, \mathsf{pp}_{\mathcal{R}_{\mathtt{ring}}}, \mathsf{comring}, \mathsf{compk}; \mathsf{b}, \mathsf{opring}, \mathsf{sk})$
7: return $\sigma = (\pi_{ring}, \mathsf{compk}, \mathsf{comring}, \pi_{eval})$
```

Security Analysis of Our First Protocol: We should first define Gen_{sign} for $\mathcal{F}_{\mathsf{rvrf}}$ and show that Gen_{sign} satisfies the anonymity defined in Definition 7 so that $\mathcal{F}_{\mathsf{rvrf}}$ gives anonymity.

Lemma 1. *Gen_{sign} in Algorithm 1 satisfies the anonymity defined in Definition 7 assuming $\mathsf{NIZK}_{\mathcal{R}_{\mathtt{ring}}}$ is ZK and Pedersen commitment is perfectly hiding.*

Proof. Assume that $\mathcal{D}$ wins the anonymity game for Gen_{sign} with an advantage ϵ. We reduce the anonymity game to a game where we remove the line 4 and change the line 6 of Algorithm 1 with $\pi_{ring} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{\mathtt{ring}}}.\mathsf{Simulate}(\mathsf{td}_{\mathcal{R}_{\mathtt{ring}}}, \mathsf{pp}_{\mathcal{R}_{\mathtt{ring}}}, \mathsf{comring}, \mathsf{compk})$ where $\mathsf{comring} = \mathsf{Com}^*(\mathsf{ring})$. Our new game is indistinguishable since $\mathsf{NIZK}_{\mathcal{R}_{\mathtt{ring}}}$ is ZK. Since in the new game, proofs are generated without the keys and compk is perfectly hiding, $\mathcal{D}$ wins the new game with probability $\frac{1}{2}$. Thus, ϵ is negligible.

Below, we give the security statement for our first construction when pk is defined as $\mathsf{Com.Commit}(x, r)$ where $\mathsf{sk} = (x, r)$ is the secret key (Alternative 1) and when pk is defined as $\mathsf{pk} = \mathsf{sk}G$ where $\mathsf{sk} = x$ (Alternative 2).

Theorem 1. *Our first protocol realizes [11,12] $\mathcal{F}_{\mathsf{rvrf}}$ running Gen_{sign} in Algorithm 1 in the random oracle model assuming that $\mathsf{NIZK}_{\mathcal{R}_{eval}}$ and $\mathsf{NIZK}_{\mathcal{R}_{ring}}$ are zero-knowledge and knowledge sound, $\mathsf{NARK}_{\mathcal{R}_{\mathsf{comring}}}$ is knowledge sound, the decisional Diffie-Hellman (DDH) problem are hard in $\mathbb{G}$ (so the CDH problem is hard as well).*

Proof. We construct a simulator Sim that simulates the honest parties in the execution of our protocol and simulates the adversary in $\mathcal{F}_{\mathsf{rvrf}}$.

– **[Simulation of keygen:]** Upon receiving $(\mathsf{keygen}, \mathsf{sid}, \mathsf{P}_i)$ from $\mathcal{F}_{\mathsf{rvrf}}$, Sim generates a secret and public key pair $\mathsf{sk} = (x, r)$ and pk by running $\mathsf{rVRF.KeyGen}$. It adds pk to a list $\mathtt{honest\_keys}$ as a key of P_i. In the end, Sim returns $(\mathsf{verificationkey}, \mathsf{sid}, \mathsf{sk}, \mathsf{pk})$ to $\mathcal{F}_{\mathsf{rvrf}}$. Sim sets $\mathsf{public\_keys}[X] = \mathsf{pk}$ and $\mathsf{secret\_keys}[X] = (x, r)$ where $X = xG$. During the simulation, Sim populates $\mathsf{public\_keys}$ with hypothetical public keys which are never revealed during the simulation or by $\mathcal{F}_{\mathsf{rvrf}}$. However, it does not populate $\mathsf{secret\_keys}$ except this part of the simulation. So, if $\mathsf{public\_keys}[X']$ is not empty for a value X' but $\mathsf{secret\_keys}[X']$ is empty, it means that Sim generated the entry $\mathsf{public\_keys}[X']$ just for the sake of the simulation with a key which is not functional as a real public key.
– **[Simulation of corruption:]** Upon receiving a message $(\mathsf{corrupted}, \mathsf{sid}, \mathsf{P}_i)$ from $\mathcal{F}_{\mathsf{rvrf}}$, Sim removes the public key pk from $\mathtt{honest\_keys}$ which is stored as a key of P_i and adds pk to $\mathtt{malicious\_keys}$.
– **[Simulation of the random oracles:]** We describe how Sim simulates the random oracles $H_{\mathbb{G}}, H, H_p$ against the real world adversaries.
Sim simulates the random oracle $H_{\mathbb{G}}$ as described in Figure 1. It selects a random element h from $\mathbb{F}_p$ for each new input and outputs hG as an output of the random oracle $H_{\mathbb{G}}$. Thus, Sim knows *the discrete logarithm of each random oracle output of* $H_{\mathbb{G}}$.
The simulation of the random oracle H is less straightforward (See Figure 2). The value W can be a preout of an input generated by a malicious party or can be an anonymous key of in generated by $\mathcal{F}_{\mathsf{rvrf}}$ for an honest party. Sim does not need to know about this but H should output $\mathtt{evaluations}[\mathsf{in}, W]$ in both cases to be consistent with $\mathcal{F}_{\mathsf{rvrf}}$. Sim treats W as if it is preout generated as in the protocol. So, Sim first obtains the discrete logarithm h of $H_{\mathbb{G}}(\mathsf{in})$ from the $H_{\mathbb{G}}$'s database and obtains $X^* = h^{-1}W$. Sim checks if $\mathsf{public\_keys}[X^*]$ exists. If it does not exist, Sim samples randomly a key pk^* which is not stored in $\mathsf{public\_keys}$ and stores $\mathsf{public\_keys}[X^*] = \mathsf{pk}^*$ just to use while sending an eval message to $\mathcal{F}_{\mathsf{rvrf}}$. Then, it sends $(\mathsf{eval}, \mathsf{sid}, \mathsf{pk}^*, W, \mathsf{in})$ to $\mathcal{F}_{\mathsf{rvrf}}$ and receives back $\mathtt{evaluations}[\mathsf{in}, W]$. Remark that if W is a pre-output generated by $\mathcal{A}$, then $\mathcal{F}_{\mathsf{rvrf}}$ matches it with the evaluation value given by $\mathcal{F}_{\mathsf{rvrf}}$. If W is an anonymous key of an honest party in the ideal world, $\mathcal{F}_{\mathsf{rvrf}}$ still returns an honest evaluation value $\mathtt{evaluations}[\mathsf{in}, W]$ even if Sim cannot know whether W is an anonymous key of an honest party in the ideal world. During the simulation of H, if $\mathcal{F}_{\mathsf{rvrf}}$ aborts, then there exists $W' \neq W$ such that $\mathtt{anonymous\_key\_map}[\mathsf{in}, W'] = \mathsf{pk}^*$. Remark that it is not possible because if it happens it means that $hX^* = W' \neq W$ where $\mathsf{public\_keys}[X^*] = \mathsf{pk}^*$, but also $W = hX^*$. Therefore, ABORT-1 never occurs.

```
Oracle $H_{\mathbb{G}}$
Input: in

if $\mathtt{H}_{\mathbb{G}}[\mathsf{in}] = \perp$
  $h \leftarrow\$ \mathbb{F}_p$
  $\mathtt{H}_{\mathbb{G}}[\mathsf{in}] := h$
  return $hG$
else:
  $h \leftarrow \mathtt{H}_{\mathbb{G}}[\mathsf{in}]$
  $\mathtt{H}_{\mathbb{G}}[\mathsf{in}] := h$
  return $hG$
```

Fig. 1. The random oracle $H_{\mathbb{G}}$

```
Oracle $H$
Input: in, $W$
if $\mathtt{H}[\mathsf{in}, W] \neq \perp$
   return $\mathtt{H}[\mathsf{in}, W]$
$h \leftarrow \mathtt{H}_{\mathbb{G}}[\mathsf{in}]$
$X^* := h^{-1}W$ // candidate commitment key
if $\mathsf{public\_keys}[X^*] = \perp$
     $\mathsf{pk}^* \leftarrow\$ \mathbb{G}$
     $\mathsf{public\_keys}[X^*] := \mathsf{pk}^*$
send $(\mathsf{eval}, \mathsf{sid}, W, \mathsf{public\_keys}[X^*], \mathsf{in})$ to $\mathcal{F}_{\mathsf{rvrf}}$
if $\mathcal{F}_{\mathsf{rvrf}}$ ignores: ABORT-1
receive $(\mathsf{evaluated}, \mathsf{sid}, W, \mathsf{in}, \mathsf{out})$ from $\mathcal{F}_{\mathsf{rvrf}}$
$\mathtt{H}[\mathsf{in}, W] := \mathsf{out}$
return $\mathtt{H}[\mathsf{in}, W]$
```

Fig. 2. The random oracle H

The simulation of the random oracle H_p (See Figure 3 for details) given the query query $(\mathsf{ad}', \mathsf{in}, \mathsf{compk}, W, R, R_m)$ makes sure that the verified signature $\sigma = (\pi_{\mathtt{eval}}, \pi_{\mathtt{ring}}, \mathsf{compk}, \mathsf{comring})$ of honest parties verifies $\pi_{\mathtt{eval}} = (c, s_1, s_2)$ via H_p as in the protocol. For this, it first parses ad' as $\mathsf{ad} \mathbin{+\!\!+} \pi_{\mathtt{ring}} \mathbin{+\!\!+} \mathsf{comring}$. If $\pi_{\mathtt{ring}}$ is verified via $\mathsf{NIZK}_{\mathcal{R}_{ring}}$, then the oracle H_p deduces that the reply to this oracle query might obtained from $\mathcal{F}_{\mathsf{rvrf}}$ in case $\mathsf{compk}, \pi_{\mathtt{ring}}, \mathsf{comring}$ are a part of a valid honest signature. If the oracle H_p obtains such verified signature σ from $\mathcal{F}_{\mathsf{rvrf}}$, it returns c if $R = s_1G + s_2K - c\,\mathsf{compk}$ and $R_m = s_1H_{\mathbb{G}}(\mathsf{in}) - cW$. We remark that if R and R_m satisfy these equalities, it means that they correspond to R and R_m generated during $\mathsf{rVRF.Verify}$ which is supposed to output 1 for the part of $\pi_{\mathtt{eval}}$.

```
Oracle $H_p$
Input: $(\mathsf{ad}', \mathsf{in}, \mathsf{compk}, W, R, R_m)$

parse $\mathsf{ad}'$ as $\mathsf{ad} \mathbin{+\!\!+} \pi_{\mathtt{ring}} \mathbin{+\!\!+} \mathsf{comring}$
if $\mathtt{H_p}[\mathsf{ad}', \mathsf{in}, \mathsf{compk}, W, R, R_m] \neq \perp$: return $\mathtt{H_p}[\mathsf{ad}', \mathsf{in}, \mathsf{compk}, W, R, R_m]$
else if  $\mathsf{NIZK}_{\mathcal{R}_{\mathtt{ring}}}.\mathsf{Verify}((\mathsf{compk}, \mathsf{comring}); \pi_{\mathtt{ring}}) \to 1$
    send $(\mathsf{request\_signatures}, \mathsf{sid}, \mathsf{ad}, W, \mathsf{in})$
    receive $(\mathsf{signatures}, \mathsf{sid}, \mathsf{in}, \mathcal{L}_\sigma)$
    if $\exists \sigma', \sigma \in \mathcal{L}_\sigma$ such that $\sigma = (\pi_{\mathtt{ring}}, \mathsf{compk}, \mathsf{comring}, ., W)$ and $\sigma' = (\pi_{\mathtt{ring}}, \mathsf{compk}, \mathsf{comring}, ., W)$
        ABORT-2
    else if $\exists \sigma \in \mathcal{L}_\sigma$ such that $\sigma = (\pi_{\mathtt{ring}}, \mathsf{compk}, \mathsf{comring}, \pi_{\mathtt{eval}}, W)$ for some $\pi_{\mathtt{eval}}$
        get $\pi_{\mathtt{eval}} = (c, s_1, s_2)$
        if $R = s_1G + s_2K - c\mathsf{compk}$, $R_m = s_1H_{\mathbb{G}}(\mathsf{in}) - cW$
            $\mathtt{H_p}[\mathsf{ad}', \mathsf{in}, \mathsf{compk}, W, R, R_m] := c$
if $\mathtt{H_p}[\mathsf{ad}', \mathsf{in}, \mathsf{compk}, W, R, R_m] = \perp$
    $c \leftarrow\$ \mathbb{F}_p$
    $\mathtt{H_p}[\mathsf{ad}', \mathsf{in}, \mathsf{compk}, W, R, R_m] := c$
return $\mathtt{H_p}[\mathsf{ad}', \mathsf{in}, \mathsf{compk}, W, R, R_m]$
```

Fig. 3. The random oracle H_p

- **[Simulation of** verify**]** Upon receiving $(\mathsf{verify}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, \mathsf{in}, \sigma)$ from the functionality $\mathcal{F}_{\mathsf{rvrf}}$, Sim runs $\mathsf{rVRF.Verify}$ algorithm of our ring VRF protocol. If it verifies, it sets $b_{\mathsf{Sim}} = 1$. Otherwise it sets $b_{\mathsf{Sim}} = 0$.
    - If $b_{\mathsf{Sim}} = 1$, it sets $X = h^{-1}W$ where $h = \mathtt{H}_{\mathbb{G}}[m]$. Then it obtains $\mathsf{pk} = \mathsf{public\_keys}[X]$ if it exists. If it does not exist, it picks a pk which is not stored in $\mathsf{public\_keys}$ and sets $\mathsf{public\_keys}[X] = \mathsf{pk}$. Then it sends $(\mathsf{verified}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, \mathsf{in}, \sigma, b_{\mathsf{Sim}}, \mathsf{public\_keys}[X])$ to $\mathcal{F}_{\mathsf{rvrf}}$ and receives back $(\mathsf{verified}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, \mathsf{in}, \sigma, \mathsf{out}, b)$.
      If $b \neq b_{\mathsf{Sim}}$, it means that the signature is not a valid signature in the ideal world, while it is in the real world. So, Sim aborts in this case (ABORT-3). If $\mathcal{F}_{\mathsf{rvrf}}$ does not verify a ring signature even if it is verified in the real world, $\mathcal{F}_{\mathsf{rvrf}}$ is in either C3-2, 1 or C3-3. If $\mathcal{F}_{\mathsf{rvrf}}$ is in C3-2, it means that $|\mathcal{W}[\mathsf{in}, \mathsf{ring}]| > |\mathsf{ring}_{mal}|$. If $\mathcal{F}_{\mathsf{rvrf}}$ is in C3-1, it means that pk belongs to an honest party but this honest party never signs in and ad for ring. So, σ is a forgery. If $\mathcal{F}_{\mathsf{rvrf}}$ is in C3- 3, it means that there exists $W' \neq W$ where $\mathtt{anonymous\_key\_map}[\mathsf{in}, W'] = \mathsf{pk}$. If $[\mathsf{in}, W']$ is stored before, it means that Sim obtained $W' = hX$ where $h = \mathtt{H}_{\mathbb{G}}[\mathsf{in}]$ but it is impossible to happen since $W = hX$.
      If $b = b_{\mathsf{Sim}}$, it sets $\mathtt{H}[\mathsf{in}, W] = \mathsf{out}$, if it is not defined before.
    - If $b_{\mathsf{Sim}} = 0$, it sets $\mathsf{pk} = \perp$ and sends $(\mathsf{verified}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, m, \sigma, b_{\mathsf{Sim}}, X)$ to $\mathcal{F}_{\mathsf{rvrf}}$. Then, Sim receives back $(\mathsf{verified}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, m, \sigma, \perp, 0)$.

We remark that ABORT-2 happens in the oracle H_p described in Figure 3 in case W is generated by $\mathcal{F}_{\mathsf{rvrf}}$ for an honest party. The reason of this is that $\mathcal{F}_{\mathsf{rvrf}}$ asks for Sim to verify or not verify all signatures with W which is not generated by $\mathcal{F}_{\mathsf{rvrf}}$. Sim runs $\mathsf{rVRF.Verify}$ for all such requests and replies accordingly. Therefore, the valid signatures for $\mathsf{in}, \mathsf{ad}'$ with malicious W (obtained via $\mathsf{request\_signatures}$) must been already validated by Sim before and $\mathtt{H_p}[(\mathsf{ad}', \mathsf{in}, \mathsf{compk}, W, R, R_m)]$ has been assigned with a random value.

We next show that the outputs of honest parties in the ideal world are indistinguishable from the honest parties running our second protocol.

Lemma 2. *Assuming that the DDH problem is hard on $\mathbb{G}$, the outputs of honest parties in our first ring VRF protocol are indistinguishable from the output of the honest parties in $\mathcal{F}_{\mathsf{rvrf}}$ running Gen_{sign} in Algorithm 1.*

Proof Sketch: The the honest evaluation outputs generated by $\mathcal{F}_{\mathsf{rvrf}}$ and generated by H in the real world protocol are in the identical distribution. The ring VRF signatures of honest parties in two worlds $((\pi_{\mathtt{ring}}, \mathsf{compk}, \mathsf{comring}, \pi_{\mathtt{eval}}, W)$ in $\mathcal{F}_{\mathsf{rvrf}}$ and $(\pi_{\mathtt{ring}}, \mathsf{compk}, \mathsf{comring}, \pi_{\mathtt{eval}}, \mathsf{preout})$ in rVRF) are in different distributions because W and preout generated differently while the rest is in an identical distribution. We can show that they are indistinguishable under the assumption that DDH problem is hard (See Appendix 4).

Next we show that the simulation executed by Sim against $\mathcal{A}$ is indistinguishable from the real protocol execution.

Lemma 3. *The view of $\mathcal{A}$ in its interaction with the simulator* Sim *is indistinguishable from the view of $\mathcal{A}$ in its interaction with real honest parties assuming that CDH is hard in $\mathbb{G}$, $H_{\mathbb{G}}, H, H_p$ are random oracles, $\mathsf{NIZK}_{\mathcal{R}_{eval}}, \mathsf{NIZK}_{\mathcal{R}_{\mathsf{ring}}}, \mathsf{NARK}_{\mathcal{R}_{\mathsf{comring}}}$ are knowledge sound and* Com *is computationally binding and perfectly hiding.*

Proof Sketch: The simulation against the real world adversary $\mathcal{A}$ is identical to the real protocol except the cases where Sim aborts. Abort-1 cannot happen as we explained. Abort-2 happens if Gen_{sign} generates the same compk for two different signatures. This happens if $\mathcal{F}_{\mathsf{rvrf}}$ selects the same compk for two different honest signatures which happens with a negligible probability. Now, we are left with the abort case (Abort-3) during the verification. For this, we show that if there exists an adversary $\mathcal{A}$ which makes Sim abort during the simulation, then we construct another adversary $\mathcal{B}$ which breaks either the CDH problem or the binding property of Com (See Appendix 4).

This completes the security proof of our first ring VRF protocol. □

5 Zero-knowledge Continuations

In this section, we describe our new notion that we call zero-knowledge (ZK) continuation. Our new notion focuses on optimizing a NIZK proving system tailored for a relation $\mathcal{R}$ such that

$$\mathcal{R} = \{(\bar{y}, \bar{z}; \bar{x}, \bar{w}_1, \bar{w}_2) : (\bar{y}, \bar{x}; \bar{w}_1) \in \mathcal{R}_1, (\bar{z}, \bar{x}; \bar{w}_2) \in \mathcal{R}_2\},$$

and $\mathcal{R}_1$, $\mathcal{R}_2$ are NP relations. At a high level, NIZK proving systems for relations as $\mathcal{R}$ are based on the commit-and-prove methodology [32,13,10] as relations $\mathcal{R}_1$ and $\mathcal{R}_2$ have input $\bar{x}$ in common. These systems typically incorporate a commitment X to $\bar{x}$ in their respective proofs or arguments for $\mathcal{R}_1$ and $\mathcal{R}_2$ to hide the witness x in $\mathcal{R}$. In our proposed NIZK for $\mathcal{R}$, we adopt a similar methodology but with a distinctive addition. Our design is specified to facilitate the efficient re-proving membership for relation $\mathcal{R}_1$ via ZK continuation. In practice, using a NIZK that ensures a ZK continuation for a subcomponent relation (i.e., in our case $\mathcal{R}_1$) means one essentially needs to create only once an expensive proof for that subcomponent relation; the initial proof can later be re-used multiple times (just after inexpensive operations), while preserving knowledge soundness and zero-knowledge of the entire NIZK. Thus, our re-used proofs stay unlinkable. Below, we formally define ZK continuation. In Section 5.1, we instantiate it via SpecialG, and finally, in section 6 we use it to instantiate our rVRF.Sign algorithm from Section 4 with fast amortised prover time.

Definition 8 (ZK Continuation). *A ZK continuation for a relation $\mathcal{R}_1$ with a vector of inputs $(\bar{y}, \bar{x})$ and witnesses $\bar{w}_1$ is a tuple of PPT algorithms* ($\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Setup}$, $\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Preprove}$, $\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Reprove}$, $\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{VerCom}$, $\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Verify}$, $\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Simulate}$) *with implicit inputs $\mathcal{R}_1$ and security parameter λ,*

– $\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Setup}(1^\lambda) \to (\mathsf{crs}, \mathsf{td}, \mathsf{pp})$: *It outputs a common reference string* crs*, a trapdoor* td *and a list* pp *of public parameters.*
– $\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Preprove}(\mathsf{crs}, \bar{y}, \bar{x}, \bar{w}_1) \to (X', \pi', b')$: *It outputs a commitment X' to $\bar{x}$ (called* opaque*), its opening b' and a proof π' constructed from vector of inputs $\bar{y}$ (called* transparent*).*
– $\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Reprove}(\mathsf{crs}, X', \pi', b') \to (X, \pi, b)$: *It outputs a new commitment X and proof π with a new opening b for the commitment.*
– $\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{VerCom}(\mathsf{pp}, X, \bar{x}, b) \to 0/1$: *It verifies that X is a commitment to $\bar{x}$ with opening b and outputs 1 if indeed that is the case and 0 otherwise.*
– $\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Verify}(\mathsf{crs}, \bar{y}, X, \pi) \to 0/1$: *It outputs 1 if it verifies and 0 otherwise.*
– $\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Simulate}(\mathsf{td}, \bar{y}) \to (\pi, X)$: *It outputs a proof π and X given a simulation trapdoor* td *and statement $(\bar{y}, \bar{x})$.*

$\mathsf{ZKCont}_{\mathcal{R}_1}$ *satisfies perfect completeness, knowledge soundness and zero-knowledge as defined below. We define perfect completeness for* Preprove *and* Reprove *separately in the most general way possible, (i.e., with inputs supplied by the adversary where possible).*
Perfect Completeness for Preprove*: For all λ, for every $(\bar{y}, \bar{x}; \bar{w}_1) \in \mathcal{R}_1$:*

$$\Pr[\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Verify}(\mathsf{crs}, \bar{y}, X, \pi) = 1 \wedge \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{VerCom}(\mathsf{pp}, X, \bar{x}, b) = 1 \mid$$
$$(\mathsf{crs}, \mathsf{td}, \mathsf{pp}) \leftarrow \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Setup}(1^\lambda), (X, \pi, b) \leftarrow \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Preprove}(\mathsf{crs}, \bar{y}, \bar{x}, \bar{w}_1)] = 1$$

Perfect Completeness for Reprove*: For all λ and PPT adversaries $\mathcal{A}$:*

$$\begin{aligned}\Pr[&(\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Verify}(\mathsf{crs}, \bar{y}, X', \pi') = 1 \ \Rightarrow \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Verify}(\mathsf{crs}, \bar{y}, X, \pi) = 1) \ \wedge \\ &\wedge \ (\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{VerCom}(\mathsf{pp}, X', \bar{x}, b') = 1 \Rightarrow \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{VerCom}(\mathsf{pp}, X, \bar{x}, b) = 1) \mid \\ &(\mathsf{crs}, \mathsf{td}, \mathsf{pp}) \leftarrow \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Setup}(1^\lambda), (\bar{y}, \bar{x}, X', \pi', b') \leftarrow \mathcal{A}(\mathsf{crs}, \mathcal{R}_1), \\ &(X, \pi, b) \leftarrow \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Reprove}(\mathsf{crs}, X', \pi', b')] = 1\end{aligned}$$

Knowledge Soundness *For all λ, for every benign auxiliary input aux (as per [4]) and every non-uniform efficient adversary $\mathcal{A}$, there exists an efficient non-uniform extractor $\mathcal{E}$ such that:*

$$\Pr[\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Verify}(\mathsf{crs}, \bar{y}, X, \pi) = 1 \wedge \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{VerCom}(\mathsf{pp}, X, \bar{x}, b) = 1 \wedge (\bar{y}, \bar{x}; \bar{w}_1) \notin \mathcal{R}_1 \mid$$
$$(\mathsf{crs}, \mathsf{td}, \mathsf{pp}) \leftarrow \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Setup}(1^\lambda), (\bar{y}, \bar{x}, X, \pi, b; \bar{w}_1) \leftarrow \mathcal{A}||\mathcal{E}(\mathsf{crs}, aux, \mathcal{R}_1)] = \mathsf{negl}(\lambda),$$

where by $(output_A; output_B) \leftarrow A||B(input)$ we denote algorithms A, B running on the same input and B having access to the random coins of A.

Finally, we introduce a new flavour of zero-knowledge property for $\mathsf{ZKCont}_{\mathcal{R}_1}$*. It allows us to formalize the concept that after an initial call to* $\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Preprove}$ *with $((\bar{y}, \bar{x}), \bar{w}_1) \in \mathcal{R}_1$, subsequent sequential calls to* $\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Reprove}$ *result in proofs that disclose no information about $\bar{x}$ or $\bar{w}_1$. Hence, the proofs obtained via sequential use of* $\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Reprove}$ *as described above are not linkable, i.e., a property targeted in the preamble of this section.*
Perfect Zero-knowledge w.r.t. $\mathcal{R}_1$: *For all λ, for every benign auxiliary input aux, for all $(\bar{y}, \bar{x}; \bar{w}_1) \in \mathcal{R}_1$, for all X', for all π', for all b', for every adversary $\mathcal{A}$, there exists a PPT algorithm* Simulate *such that:*

$$\begin{aligned}&\Pr[\mathcal{A}(\mathsf{crs}, aux, \pi, X, \mathcal{R}_1) = 1 \mid (\mathsf{crs}, \mathsf{td}, \mathsf{pp}) \leftarrow \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Setup}(1^\lambda), \\ &\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Verify}(\mathsf{crs}, \bar{y}, X', \pi') = 1, (\pi, X, \_) \leftarrow \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Reprove}(\mathsf{crs}, X', \pi', b')] \\ =&\Pr[A(\mathsf{crs}, aux, \pi, X, \mathcal{R}_1) = 1 \mid (\mathsf{crs}, \mathsf{td}, \mathsf{pp}) \leftarrow \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Setup}(1^\lambda), \\ &\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Verify}(\mathsf{crs}, \bar{y}, X', \pi') = 1, (\pi, X) \leftarrow \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Simulate}(\mathsf{td}, \bar{y})]\end{aligned}$$

5.1 Specialised Groth16 Proofs

Below we instantiate our ZK continuation notion with a scheme that we call $\mathsf{SpecialG}$. It is based on Groth16 zkSNARK [29]. As in [29], we use a standard quadratic arithmetic program $\mathcal{Q}$ (QAP) of size m defined over field $\mathbb{F}_q$ (See Appendix D for more details). Then given $\mathcal{Q}$, we then set $\mathcal{R}_\mathcal{Q}$ that corresponds to $\mathcal{R}_1$. $\mathcal{R}_\mathcal{Q}$ consists of pairs $((\bar{y}, \bar{x}); \bar{w}) \in \mathbb{F}_q^l \times \mathbb{F}_q^{n-l} \times \mathbb{F}_q^{m-n}$ where $\mathbb{F}_q$ is a field.
We let $\mathbf{G}$ be a pairing friendly elliptic curve with an efficient and non-degenerate pairing e. We denote its first and second source groups by $\mathbb{G}_1, \mathbb{G}_2$ with generators G_1 and G_2, respectively. Given a vector $\vec{x}$ of field elements and a group element $G \in \mathbb{G}_1$ or $\mathbb{G}_2$, we use short hand notation $\vec{x} \cdot G$ to naturally represent the corresponding vector of group elements. $\mathsf{SpecialG}$ for relation $\mathcal{R}_\mathcal{Q}$ works as follows:

- $\mathsf{SpecialG.Setup}(1^\lambda, \mathcal{R}_\mathcal{Q}) \rightarrow (\mathsf{crs}, \mathsf{td}, \mathsf{pp})$: It is identical to the LegoSNARK $\mathsf{ccGro16}$ [10, Fig. 22] setup which is an extension of original Groth16 [29] setup by two additional group elements in crs and one field element in td which are underlined next. td consists of $\alpha, \beta, \gamma, \delta, \tau, \underline{\eta} \leftarrow\$ \mathbb{F}_q^*$ and $\mathsf{crs} = (\bar{\sigma}_1, \bar{\sigma}_2)$ where $\bar{\sigma}_1 = (\alpha, \beta, \delta, \{\tau_i\}_{i=0}^{d-1}, \{\gamma_i\}_{i=1}^{n}, \underline{\frac{\eta}{\gamma}}, \{\delta_i\}_{i=n+1}^{m}, \{\frac{1}{\delta}\tau^i t(\sigma)\}_{i=0}^{d-2}, \underline{\frac{\eta}{\delta}}) \cdot G_1, \bar{\sigma}_2 = (\beta, \gamma, \delta, \{\tau^i\}_{i=0}^{d-1}) \cdot G_2$. Here, $\gamma_i = \frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\gamma}$ and $\delta_i = \frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\delta}\}_{i=n+1}^{m}$. We let $\mathsf{pp} = (\{K_i\}_{\ell+1}^{n}, K_\gamma) = (\{\gamma_i G_1\}_{i=l+1}^{n}, \frac{\eta}{\gamma} G_1)$ and $K_\delta = \frac{\eta}{\delta} \cdot G_1$.
  A Groth16 proof for $\mathcal{R}_\mathcal{Q}$ needs the public statement $(\bar{y}, \bar{x})$ for verification. Differently than this, we want to achieve the verification of a Groth16 proof without $\bar{x}$ but with the commitment to $\bar{x}$. Therefore, we need additional elements in crs to be able to still execute the verification.
- $\mathsf{SpecialG.Preprove}(\mathsf{crs}, \bar{y}, \bar{x}, \bar{w}_1) \leftarrow (X', \pi', b')$: It runs the proving algorithm of Groth16 SNARK outputting $\pi' = (A, B, C) \in (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_1)$, then it lets $b' = 0$, computes the deterministic commitment $X' = \sum_{i=\ell+1}^{n} x_i K_i$ to $\bar{x}$. In more detail,

$$b' = 0; r, s \xleftarrow{\$} \mathbb{F}_p; X' = \sum_{i=l+1}^{n} v_i \cdot \frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\gamma} \cdot G_1;$$

$$a = \alpha + \sum_{i=0}^{m} v_i \cdot a_i(\tau) + r \cdot \delta; b = \beta + \sum_{i=0}^{m} v_i \cdot b_i(\tau) + s \cdot \delta;$$

$$c = \frac{\sum_{i=n+1}^{m} (v_i(\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau))) + h(\tau)t(\tau)}{\delta} + o \cdot s + u \cdot r - r \cdot s \cdot \delta;$$

$$\pi' = (A, B, C) = (aG_1, uG_2, vG_1),$$

  where $\bar{v} = (1, x_1, \ldots, x_n, w_1, \ldots, w_{m-n})$, $\bar{y} = (x_1, \ldots, x_l)$, $\bar{x} = (x_{l+1}, \ldots, x_n)$, $\bar{w} = (w_1, \ldots, w_{m-n})$ (same as per definition of QAP).
- $\mathsf{SpecialG.Reprove}(\mathsf{crs}, X', \pi', b') \leftarrow (X, \pi, b)$: It rerandomizes π and the commitment X by using b' i.e., given $\pi' = (A', B', C')$, pick $b, r_1, r_2 \xleftarrow{\$} \mathbb{F}_p$ and let $X = X' + (b - b')K_\gamma, \pi = (A, B, C)$ where $A = \frac{A'}{r_1}, B = r_1 B' + r_1 r_2 \delta G_2, C = C' + r_2 A' - (b - b')K_\delta$.
  The $\mathsf{Preprove}$ and $\mathsf{Reprove}$ procedures of $\mathsf{SpecialG}$ are identical to the proving procedure in $\mathsf{ccGro16}$. In $\mathsf{SpecialG}$, we split these procedures because we aim to run $\mathsf{Preprove}$ once which contains heavier operations and then we can efficiently run $\mathsf{Reprove}$ multiple times with lighter operations.
  The next algorithms $\mathsf{SpecialG.VerCom}$ and $\mathsf{SpecialG.Verify}$ are identical to $\mathsf{ccGro16}$ commitment and proof verification algorithms, respectively.
- $\mathsf{SpecialG.VerCom}(\mathsf{pp}, X, \bar{x}, b) \rightarrow 0/1$: It outputs 1 iff $X = \sum_{i=l+1}^{n} x_i K_i + bK_\gamma$.
- $\mathsf{SpecialG.Verify}(\mathsf{crs}, \bar{y}, X, \pi) \rightarrow 0/1$: It outputs 1 iff the following holds $e(A, B) = e(\alpha G_1, \beta G_2) + e(X + Y, \gamma G_2) + e(C, \delta G_2)$ where $\pi = (A, B, C)$, $Y = \sum_{i=1}^{l} x_i \delta_i G_1$ and $\bar{y} = (x_1, \ldots, x_l)$.
  We remark that $\mathsf{SpecialG.Verify}$ corresponds to the verify algorithm of Groth16 zkSNARK when X is the output of $\mathsf{SpecialG.Preprove}$.

– $\mathsf{SpecialG.Simulate}(\mathsf{td}, \bar{y}, \mathcal{R}_{\mathcal{Q}}) \to (\pi, X)$: It samples $\tilde{x}, a, b \xleftarrow{\$} \mathbb{F}_p$ and let $\pi = (aG_1, bG_2, cG_1)$ where $c = \frac{ab - \alpha\beta - \sum_{i=1}^{l} x_i(\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)) - \tilde{x}}{\delta}$ and, by definition $\bar{y} = (x_1, \ldots, x_l)$. Note that π is a simulated proof for transparent input $\bar{y}$ and commitment $X = \tilde{x}G_1$.

Next, we prove that $\mathsf{SpecialG}$ is a zero-knowledge continuation. We show that the knowledge soundness property of $\mathsf{SpecialG}$ (i.e., as defined for $\mathsf{ZKCont}_{\mathcal{R}_1}$) is implied by the knowledge soundness property of commit-carrying SNARK with double binding (cc-SNARK with double binding, see Definition 3.4 [10]); our notion of zero-knowledge for $\mathsf{ZKCont}_{\mathcal{R}_1}$ is, in fact a new and stronger notion so we prove that directly. Formally, we have:

Theorem 2. *Let $\mathcal{R}_{\mathcal{Q}}$ be a relation as related to a QAP such that additionally $\{a_k(X)\}_{k=0}^{n}$ are linearly independent polynomials. Then, in the AGM [23],* $\mathsf{SpecialG}$ *is a zero-knowledge continuation as per Definition 8.*

Proof. It is straightforward to show that $\mathsf{SpecialG}$ has perfect completeness for $\mathsf{Preprove}$ thanks to the completeness of the Groth16 zkSNARK because $\mathsf{SpecialG.Verify}$ is the same as the verification algorithm of the Groth16 zkSNARK when X is the output of $\mathsf{Preprove}$. For the perfect completeness for $\mathsf{Reprove}$, we have $X', \pi' = (A', B', C')$ and b' given by the adversary and $X, \pi = (A, B, C), b$ generated by $\mathsf{Reprove}(crs, X', \pi', b')$. Clearly, if $\mathsf{SpecialG.VerCom}(pp, X', \bar{x}, b')$ verifies, $\mathsf{SpecialG.VerCom}(pp, X, \bar{x}, b)$ verifies. The verification of X, π given that X', π' is verified becomes straightforward when we replace X with $X' + (b - b')K_\gamma$, A with $\frac{A'}{r_1}$, B with $r_1B' + r_1r_2\delta G_2$ C with $= C' + r_2A' - (b - b')K_\delta$. After the replacement, the equality check reduced to $e(A', B') = e(\alpha G_1, \beta G_2) + e(X' + Y, \gamma G_2) + e(C', \delta G_2)$ which is the verification check for X', π'. So, if the adversarial proof verifies then the proof generated by $\mathsf{Reprove}$ verifies as desired from the perfect completeness property of $\mathsf{Reprove}$.

We next prove knowledge-soundness (KS) an in Definition 8 by first arguing $\mathsf{SpecialG}$ is a cc-SNARK with double binding (see Definition 3.4 [10]). We use the fact that $\mathsf{ccGro16}$ as defined by the NILP detailed in Fig.22, Appendix H.5 [10] satisfies that latter definition. Moreover, $\mathsf{SpecialG}$'s Setup on one hand, and $\mathsf{ccGro16}$'s KeyGen, on the other hand, are the same procedure. Also $\mathsf{SpecialG}$ and $\mathsf{ccGro16}$ share the same verification algorithm. Hence, translating the notation appropriately, $\mathsf{SpecialG}$ also satisfies KS of a cc-SNARK with double binding.

Let $\mathcal{A}_{\mathsf{SpecialG}}$ be an adversary for KS in Definition 8 and define adversary $\mathcal{A}_{\mathsf{ccGro16}}$ for KS in Definition 3.4 [10]:

$$\begin{aligned}&\textit{If } \mathcal{A}_{\mathsf{SpecialG}}(\mathsf{crs}, \mathsf{pp}, aux, \mathcal{R}_{\mathcal{Q}}) \textit{ outputs } (\bar{y}, \bar{x}, X, \pi, b)\\ &\qquad \textit{then } \mathcal{A}_{\mathsf{ccGro16}}(\mathsf{crs}, aux, \mathcal{R}_{\mathcal{Q}}) \textit{ outputs } (\bar{y}, X, \pi).\end{aligned}$$

Given extractor $\mathcal{E}_{\mathsf{ccGro16}}$ fulfilling Definition 3.4 [10] for $\mathcal{A}_{\mathsf{ccGro16}}$, we construct extractor $\mathcal{E}_{\mathsf{SpecialG}}$ for $\mathcal{A}_{\mathsf{SpecialG}}$

$$\begin{aligned}&\textit{If}\, \mathcal{E}_{\mathsf{ccGro16}}(\mathsf{crs}, aux, \mathcal{R}_{\mathcal{Q}}) \textit{ outputs } (\bar{x}^*, b^*, \bar{w}^*)\\ &\qquad \textit{then } \mathcal{E}_{\mathsf{SpecialG}}(\mathsf{crs}, aux, \mathcal{R}_{\mathcal{Q}}) \textit{ outputs } \bar{w}^*;\\ &\textit{Otherwise } \mathcal{E}_{\mathsf{ccGro16}}(\mathsf{crs}, aux, \mathcal{R}_{\mathcal{Q}}) \textit{ outputs } \perp.\end{aligned}$$

We show $\mathcal{E}_{\mathsf{SpecialG}}$ fulfils Definition 8 for $\mathcal{A}_{\mathsf{SpecialG}}$. Assume by contradiction that is not the case. This implies there exists an auxiliary input aux such that each:

$$\mathsf{SpecialG.Verify}(\mathsf{crs}, \bar{y}, X, \pi, \mathcal{R}_{\mathcal{Q}}) = 1 \quad (10), \ \mathsf{SpecialG.VerCom}(\mathsf{pp}, X, \bar{x}, b) = 1 \quad (20), \ (\bar{y}, \bar{x}; \bar{w}) \notin \mathcal{R}_{\mathcal{Q}} \quad (30)$$

holds with non-negligible probability. Since (20) holds with non-negligible probability and verification (for both proofs and commitments actually) is identical in $\mathsf{SpecialG}$ and $\mathsf{ccGro16}$ respectively, and since $\mathcal{E}_{\mathsf{ccGro16}}$ is an extractor for $\mathcal{A}_{\mathsf{ccGro16}}$ as per Definition 3.4 [10], then each of the two events

$\mathsf{ccGro16}.\mathit{VerCommit}^*(ck, X, \bar{x}^*, b^*) = 1$ (40), $(\bar{y}, \bar{x}^*; \bar{w}^*) \in \mathcal{R}_{\mathcal{Q}}$ (50) holds with overwhelming probability. Since (20) holds with non-negligible probability and (40) holds with overwhelming probability and together with (ii) from Definition 3.4 [10] we obtain that $\bar{x}^* = \bar{x}$. Since (50) holds with overwhelming probability, it implies $(\bar{y}, \bar{x}; \bar{w}^*) \in \mathcal{R}_{\mathcal{Q}}$ with overwhelming probability which contradicts our assumption, so our claim that $\mathsf{SpecialG}$ does not have KS as per Definition 8 is false.

Finally, regarding zero-knowledge, it is clear that if $\pi = (A, B, C)$ is part of the output of $\mathsf{SpecialG.Reprove}$, then A and B are uniformly distributed as group elements in their respective groups. This holds, as long as the input to $\mathsf{SpecialG.Reprove}$ is a verifying proof, even when the proof was maliciously generated. Hence, it is easy to check that the output π' of $\mathsf{SpecialG.Simulate}$ is identically distributed to a proof π output by $\mathsf{SpecialG.Reprove}$ so the perfect zero-knowledge property holds for $\mathsf{SpecialG}$.

5.2 Putting Together a NIZK and a ZKCont for Proving $\mathcal{R}$

Let $\mathsf{ZKCont}_{\mathcal{R}_1}$ be a zk continuation for $\mathcal{R}_1$ (from preamble of this section) with public parameter pp and let $\mathsf{NIZK}_{\mathcal{R}'_2}$ be a NIZK for $\mathcal{R}'_2$ defined as

$$\mathcal{R}'_2 = \{(X, \bar{z}; \bar{x}, b, \bar{w}_2) : \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{VerCom}(\mathsf{pp}, X, \bar{x}, b) = 1 \ \wedge \ (\bar{z}, \bar{x}; \bar{w}_2) \in \mathcal{R}_2\},$$

with $\mathcal{R}_2$ from preamble of Section 5. Then we define the system $\mathsf{NIZK}_{\mathcal{R}}$ for relation $\mathcal{R}$ from the preamble of this section as:

- $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Setup}(1^\lambda) \to (\mathsf{crs}_{\mathcal{R}} = (\mathsf{crs}, crs_{\mathcal{R}'_2}), \mathsf{td}_{\mathcal{R}} = (\mathsf{td}, \mathsf{td}_{\mathcal{R}'_2}), \mathsf{pp}_{\mathcal{R}} = \mathsf{pp})$: Here, $(\mathsf{crs}, \mathsf{td}, \mathsf{pp}) \leftarrow \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Setup}(1^\lambda, \mathcal{R}_1)$, $(crs_{\mathcal{R}'_2}, \mathsf{td}_{\mathcal{R}'_2}) \leftarrow \mathsf{NIZK}_{\mathcal{R}'_2}.\mathsf{Setup}(1^\lambda)$.
- $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Prove}(\mathsf{crs}_{\mathcal{R}}, \bar{y}, \bar{z}; \bar{x}, \bar{w}_1, \bar{w}_2) \to (\pi_1, \pi_2, X)$: Here (X', π'_1, b') is generated by $\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Preprove}(\mathsf{crs}, \bar{y}, \bar{x}, \bar{w}_1)$ and (X, π_1, b) generated by $\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Reprove}(\mathsf{crs}, X', \pi'_1, b')$ and π_2 is generated by $\mathsf{NIZK}_{\mathcal{R}'_2}.\mathsf{Prove}(crs_{\mathcal{R}'_2}, X, \bar{z}; \bar{x}, b, \bar{w}_2)$.
- $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Verify}(\mathsf{crs}_{\mathcal{R}}, (\bar{y}, \bar{z}), (\pi_1, \pi_2, X)) \to 0/1$: It outputs 1 iff $\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Verify}(\mathsf{crs}, \bar{y}, X, \pi_1) = 1$ and $\mathsf{NIZK}_{\mathcal{R}'_2}.\mathsf{Verify}(crs_{\mathcal{R}'_2}, X, \bar{z}, \pi_2) = 1$.
- $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Simulate} : (\mathsf{td}_{\mathcal{R}}, \bar{y}, \bar{z}) \mapsto (\pi_1, \pi_2, X)$ where $(\pi_1, X) \leftarrow \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Simulate}(\mathsf{td}, \bar{y})$, $\pi_2 \leftarrow \mathsf{NIZK}_{\mathcal{R}'_2}.\mathsf{Simulate}(\mathsf{td}_{\mathcal{R}'_2}, X, \bar{z})$.

Theorem 3. *If $\mathsf{ZKCont}_{\mathcal{R}_1}$ is a zk continuation for $\mathcal{R}_1$ and $\mathsf{NIZK}_{\mathcal{R}'_2}$ is a NIZK for $\mathcal{R}'_2$ for some appropriately chosen public parameters pp, then the $\mathsf{NIZK}_{\mathcal{R}}$ construction described above is a NIZK for $\mathcal{R}$.*

Proof sketch: The correctness, knowledge soundness and zk properties of $\mathsf{NIZK}_{\mathcal{R}}$ comes from the same properties of $\mathsf{ZKCont}_{\mathcal{R}_1}$ and $\mathsf{NIZK}_{\mathcal{R}'_2}$. See Appendix B for the proof.

6 Our Second Ring VRF Construction based on ZKCont

We enhance our construction from Section 4 by incorporating ZKCont. This protocol leverages the rerandomization properties of ZKCont, allowing a signer to generate a signature for a different in within the same ring without having to recompute the most expensive part of the NIZK proof related to the key membership of the ring. $\mathsf{CommitRing}$, $\mathsf{OpenRing}$ works as in the first ring VRF protocol and KeyGen works as in the alternative-1 of our first protocol.

- $\mathsf{rVRF.Setup}(1^\lambda)$ outputs $\mathsf{pp}_{rvrf} = (crs_{\mathcal{R}^{\mathtt{inner}}_{\mathtt{ring}}}, crs_{\mathsf{comring}}, \mathsf{pp}_{\mathcal{R}^{\mathtt{inner}}_{\mathtt{ring}}}, (p, \mathbb{G}, G, K), \mathbb{F}_p)$ where $\mathsf{crs}_{\mathcal{R}^{\mathtt{inner}}_{\mathtt{ring}}}$ and $\mathsf{pp}_{\mathcal{R}^{\mathtt{inner}}_{\mathtt{ring}}}$ are generated by $\mathsf{ZKCont.Setup}(1^\lambda, \mathcal{R}^{\mathtt{inner}}_{\mathtt{ring}})$ and crs_{comring} is generated by $\mathsf{NARK}_{\mathcal{R}_{\mathsf{comring}}}.\mathsf{Setup}(1^\lambda)$.

– $\mathsf{rVRF.Sign}(\mathsf{sk}, \mathsf{comring}, \mathsf{opring}, \mathsf{in}, \mathsf{ad})$ computes $\mathsf{preout} = \mathsf{sk}H_{\mathbb{G}}(\mathsf{in})$, lets $\mathsf{out} = H(\mathsf{in}, \mathsf{preout})$. Then runs $\mathsf{NIZK}_{\mathcal{R}_{\mathtt{rvrf}}}.\mathsf{Prove}(\mathsf{comring}, \mathsf{preout}, \mathsf{in}, \mathsf{out}; \mathsf{sk}, \mathsf{opring}) \to \pi_{rvrf}$ where

$$\mathcal{R}_{\mathtt{rvrf}} = \left\{ \begin{matrix} \mathsf{comring}, (\mathsf{preout}, \mathsf{in}, \mathsf{out}); \\ x, r, \mathsf{opring} \end{matrix} \middle| \begin{matrix} (\mathsf{comring}, x; r, \mathsf{opring}) \in \mathcal{R}^{\mathtt{inner}}_{\mathtt{ring}}, \\ (\mathsf{preout}, \mathsf{in}, \mathsf{out}; x) \in \mathcal{R}_{\mathsf{out}} \end{matrix} \right\} \text{ and}$$

and

$$\mathcal{R}^{\mathtt{inner}}_{\mathtt{ring}} = \left\{ (\mathsf{comring}, x; r, \mathsf{opring}) \middle| \begin{matrix} \mathsf{pk} = \mathsf{rVRF.OpenRing}(\mathsf{comring}, \mathsf{opring}), \\ x = \mathsf{Com.Open}(\mathsf{pk}; x, r) \end{matrix} \right\},$$

$$\mathcal{R}_{\mathsf{out}} = \{((\mathsf{preout}, \mathsf{in}, \mathsf{out}), \mathsf{sk}; \bot) : \mathsf{preout} = \mathsf{sk}H_{\mathbb{G}}(\mathsf{in}), \mathsf{out} = H(\mathsf{in}, \mathsf{preout})\}$$

We instantiate $\mathsf{NIZK}_{\mathcal{R}_{\mathtt{rvrf}}}.\mathsf{Prove}$ as described in Section 5.2 where $\mathcal{R}_1 = \mathcal{R}^{\mathtt{inner}}_{\mathtt{ring}}$ and $\mathcal{R}'_2 = \mathcal{R}_{eval}$ and $\mathcal{R} = \mathcal{R}_{\mathtt{rvrf}}$. It works as follows: It runs $\mathsf{ZKCont}_{\mathcal{R}^{\mathtt{inner}}_{\mathtt{ring}}}.\mathsf{Preprove}(crs_{\mathcal{R}^{\mathtt{inner}}_{\mathtt{ring}}}, \mathsf{comring}, x, r, \mathsf{opring})$ and obtains $\mathsf{compk}', \pi', \mathsf{b}'$. Then, it runs $\mathsf{ZKCont}_{\mathcal{R}^{\mathtt{inner}}_{\mathtt{ring}}}.\mathsf{Reprove}(crs_{\mathcal{R}^{\mathtt{inner}}_{\mathtt{ring}}}, X', \pi', \mathsf{b}')$ and obtains $(\mathsf{compk}, \pi_{\mathtt{inner}}, \mathsf{b})$. Finally, it lets $\mathsf{preout} = xH_{\mathbb{G}}(\mathsf{in})$ and runs $\mathsf{NIZK}_{\mathcal{R}_{eval}}.\mathsf{Prove}(\mathsf{compk}, \mathsf{preout}, \mathsf{in}; x, \mathsf{b}) \to \pi_{\mathtt{eval}}$ as described in Section 4 with $\mathsf{ad}' = \mathsf{ad} \mathbin{+\!\!+} \pi_{\mathtt{ring}} \mathbin{+\!\!+} \mathsf{comring}$. In the end, it returns the ring VRF signature $\sigma = (\mathsf{compk}, \pi_{\mathtt{inner}}, \mathsf{comring}, \pi_{\mathtt{eval}}, \mathsf{preout})$.

– $\mathsf{rVRF.Verify} : (\mathsf{comring}, \mathsf{in}, \mathsf{ad}, \sigma) \mapsto (1, \mathsf{out}) \vee (0, \bot)$ it parses σ as $(\mathsf{compk}, \pi_{\mathtt{inner}}, \mathsf{comring}, \pi_{\mathtt{eval}}, \mathsf{preout}$ and runs $\mathsf{NIZK}_{\mathcal{R}_{\mathtt{rvrf}}}.\mathsf{Verify}(\mathsf{comring}, \mathsf{preout}, \mathsf{in}, \mathsf{out}; \pi_{\mathtt{eval}}, \mathsf{compk}, \pi_{\mathtt{inner}})$ i.e., runs $\mathsf{ZKCont}_{\mathcal{R}^{\mathtt{inner}}_{\mathtt{ring}}}.\mathsf{Verify}(crs_{\mathcal{R}^{\mathtt{inner}}_{\mathtt{ring}}}, \mathsf{comring}, \mathsf{compk}, \pi_{\mathtt{inner}})$ and $\mathsf{NIZK}_{\mathcal{R}_{eval}}.\mathsf{Verify}(\mathsf{compk}, \mathsf{preout}, \mathsf{in}; \pi_{\mathtt{eval}})$. If all verify, it outputs $(1, \mathsf{out} = H(\mathsf{in}, \mathsf{preout}))$. Otherwise, it returns $(0, \bot)$.

Theorem 4. *Our specialized* rVRF *realizes* $\mathcal{F}_{\mathsf{rvrf}}$ *running* Gen_{sign} *(Algorithm 2) [11,12] in the random oracle model assuming that* ZKCont *is zero-knowledge and knowledge sound as defined in Definition 8 and* $\mathsf{NIZK}_{\mathcal{R}_{eval}}$ *is zero-knowledge and knowledge sound,* $\mathsf{NARK}_{\mathcal{R}_{\mathsf{comring}}}$ *is knowledge sound, the DDH problem are hard in* $\mathbb{G}$ *(so the CDH problem is hard as well) and the commitment scheme* Com *is binding and perfectly hiding.*

Algorithm 2 $\mathsf{Gen}_{sign}(\mathsf{ring}, \mathsf{sk} = (x, r), \mathsf{pk}, \mathsf{ad}, \mathsf{in})$

```
1: $\mathsf{comring}, \mathsf{opring} \leftarrow \mathsf{rVRF.CommitRing}(\mathsf{ring}, \mathsf{pk})$
2: $\mathsf{compk}', \pi', \mathsf{b}' \leftarrow \mathsf{ZKCont.Preprove}(crs, \mathsf{comring}, \mathsf{sk}, (r, \mathsf{opring}))$
3: $\mathsf{compk}, \pi_{\mathtt{inner}}, \mathsf{b} \leftarrow \mathsf{ZKContReprove}(crs, \mathsf{compk}', \pi', \mathsf{b}')$
4: $c, s_1, s_2 \leftarrow\!\!\$\ \mathbb{F}_p$
5: $\pi_{eval} \leftarrow (c, s_1, s_2)$
6: return $\sigma = (\mathsf{compk}, \pi_{\mathtt{inner}}, \mathsf{comring}, \pi_{\mathtt{eval}})$
```

Proof Sketch: The security proof follows very similar to our first construction. We construct the same Sim described in the proof of Theorem 1 because in the second construction, random oracles in this construction are the same as in the first contruction. Then, we use the result of Lemma 1 because preout is the same. The only slight difference is in Lemma 3 since Gen_{sign} is different than Algorithm 1. There, we run the simulator and extractor of $\mathsf{NIZK}_{\mathcal{R}_{\mathtt{rvrf}}}$ instead of extractors of $\mathsf{NIZK}_{\mathcal{R}_{\mathtt{ring}}}$ and $\mathsf{NIZK}_{\mathcal{R}_{eval}}$. See Appendix A.2 for more details.

7 Conclusion

We introduced a novel cryptographic primitive ring VRF in this paper which combines the unique properties of VRFs and ring signatures. Our new primitive has notable use cases in identity systems, where users can register their public keys and generate pseudonyms using Ring VRF outputs, ensuring privacy protection while preventing Sybil behaviour. Ring VRF finds applications in a wide range of other cases, including rate limiting systems, rationing, and leader elections. We presented two distinct Ring VRF constructions, one offering flexibility in instantiation and the other focusing on optimizing signature generation within the same ring. Moreover, we introduced the notion of ZK continuations enabling the efficient regeneration of proofs by preserving the ZK property.

Instantiation of our second protocol with $\mathsf{SpecialG}$*:* Since $\mathsf{SpecialG}$ is ZKCont, we can instantiate our second protocol with $\mathsf{SpecialG}$. In this instantiation, we let $\mathbb{G} = \mathbb{G}_1$ generated in $\mathsf{SpecialG.Setup}$. We present an appropriate $\mathsf{Com.Commit(sk)}$ algorithm that together with $\mathsf{SpecialG}$ efficiently instantiate the NIZK for $\mathcal{R}_{\mathtt{ring}}^{\mathtt{inner}}$. To make this efficiently provable inside the SNARK, we use the Jubjub Edwards curve $\mathbb{J}$ which contains a large subgroup $\mathbf{J}$ of prime order $p_{\mathbf{J}}$. Here, $p_{\mathbf{J}} < p$ where p is the order of $\mathbb{G}$ used in our ring VRF construction. We let $J_0, J_1, J_2 \in \mathbf{J}$ be independent generators. We also fix a parameter κ where $(\log_2 p)/2 < \kappa < \log_2 p_{\mathbf{J}}$. $\mathsf{Com.Commit(sk)}$ first samples $\mathsf{sk}_1, \mathsf{sk}_2 \in 2^{\kappa}$ where $\mathsf{sk} = \mathsf{sk}_0 + \mathsf{sk}_1\, 2^{\lambda} \mod p$ and samples a blinding factor $d \leftarrow\!\!\$\ \mathbb{F}_{p_{\mathbf{J}}}$. In the end, it outputs $\mathsf{sk}_0, \mathsf{sk}_1, d$ as an opening and the commitment $\mathsf{pk} = \mathsf{sk}_0\, J_0 + \mathsf{sk}_1\, J_1 + dJ_2$ as a public key of our ring VRF construction. This commitment scheme is binding and perfectly hiding as our ring VRF construction requires because pk is, in fact, a Pedersen commitment. Indeed, pk is a Pedersen commitment to sk because we can represent $\mathsf{sk} = \mathsf{sk}_0\, J_0 + \mathsf{sk}_1 \mod p$ since we have selected κ accordingly.

We can instantiate Com^* with a Merkle tree hash function by setting the leaves as the public keys of the ring. Then, we instantiate $\mathsf{NARK}_{\mathcal{R}_{\mathsf{comring}}}\mathsf{.Prove}$ with inclusion proof of a key with respect to the Merkle tree root $\mathsf{comring}$.

In this case, the first run of $\mathsf{rVRF.Sign}$ for ring with $\mathsf{SpecialG}$ runs linear time in terms of the size of the statement and the witness as in the Groth16 zkSNARK [29] because it runs $\mathsf{SpecialG.Preprove}$ and $\mathsf{SpecialG.Reprove}$. Since the size of opring is $O(\log n)$, the first run of $\mathsf{rVRF.Sign}$ for a ring with $\mathsf{SpecialG}$ is $O(\log n)$. For the next signatures for the same ring, $\mathsf{rVRF.Sign}$ runs only $\mathsf{SpecialG.Reprove}$ which is 4 multiplications in $\mathbb{G}_1$ and 2 multiplications in $\mathbb{G}_2$ and $\mathsf{NIZK}_{\mathcal{R}_{eval}}\mathsf{.Prove}$ which need 3 multiplications in $\mathbb{G}_1$. The proving time becomes constant after first signing. The verification time is $O(1)$ because $\mathsf{comring}$ has a constant size. We note that if we did not deploy a Merkle tree hash function for $\mathsf{comring}$ and let $\mathsf{comring} = \mathsf{ring}$, the signing the first signature and verification times would be $O(n)$. So, $\mathsf{CommitRing}$ optimizes the the signing and verification times.

Instantiation of our first protocol: Our instantiation commits to the ring using KZG commitments (i.e., $\mathsf{Com}^*\mathsf{.Commit}$) to the x and y coordinates of the public keys. One can design a simple constraint system to verify the correctness of such a commitment (i.e., $\mathsf{rVRF.OpenRing}$) inside the custom SNARK for $\mathcal{R}_{\mathtt{ring}}$ as in [15] without additional cost, but modified to obtain zero-knowledge [26]. For this protocol, the prover needs to know the entire ring, i.e. opring is the entire ring rather than a KZG opening, which results in $O(n \log n)$ proving time unlike in the second protocol but the verification time is constant. Even though, this instantiation does not allow fast reproving, it is concretely fast with proving time under a second for rings of size up to a few thousand (comparable to the benchmarks in [15]) without needing opening constraints inside the SNARK.

References

1. Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 913–930. ACM, 2018.

2. Christian Badertscher, Peter Gaži, Iñigo Querejeta-Azurmendi, and Alexander Russell. On uc-secure range extension and batch verification for ecvrf. *Cryptology ePrint Archive*, 2022.
3. Adam Bender, Jonathan Katz, and Ruggero Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. In *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3*, pages 60–79. Springer, 2006.
4. Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, 2014.
5. Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, Jens Groth, and Christophe Petit. Short accountable ring signatures based on ddh. In *Computer Security–ESORICS 2015: 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part I*, pages 243–265. Springer, 2016.
6. Maria Borge, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, and Bryan Ford. Proof-of-personhood: Redemocratizing permissionless cryptocurrencies. In *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 23–26, 2017.
7. Joakim Brorsson, Bernardo David, Lorenzo Gentile, Elena Pagnin, and Paul Stankovski Wagner. Papr: Publicly auditable privacy revocation for anonymous credentials. In *Cryptographers' Track at the RSA Conference*, pages 163–190. Springer, 2023.
8. Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 201–210, 2006.
9. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology—EUROCRYPT 2001: International Conference on the Theory and Application of Cryptographic Techniques Innsbruck, Austria, May 6–10, 2001 Proceedings 20*, pages 93–118. Springer, 2001.
10. Matteo Campanelli, Dario Fiore, and Anaïs Querol. Legosnark: Modular design and composition of succinct zero-knowledge proofs. Cryptology ePrint Archive, Paper 2019/142, 2019. `https://eprint.iacr.org/2019/142`.
11. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. `https://eprint.iacr.org/2000/067`.
12. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 2001 IEEE International Conference on Cluster Computing*, pages 136–145. IEEE, 2001.
13. Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*, 2003.
14. Nishanth Chandran, Jens Groth, and Amit Sahai. Ring signatures of sub-linear size without random oracles. In *Automata, Languages and Programming: 34th International Colloquium, ICALP 2007, Wrocław, Poland, July 9-13, 2007. Proceedings 34*, pages 423–434. Springer, 2007.
15. Oana Ciobotaru, Fatemeh Shirazi, Alistair Stewart, and Sergey Vasilyev. Accountable light client systems for PoS blockchains. *Cryptology ePrint Archive*, 2022. `https://eprint.iacr.org/2022/1205`.
16. Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 66–98. Springer, 2018.
17. Vanesa Daza, Abida Haque, Alessandra Scafuro, Alexandros Zacharakis, and Arantxa Zapico. Mutual accountability layer: accountable anonymity within accountable trust. In *International Symposium on Cyber Security, Cryptology, and Machine Learning*, pages 318–336. Springer, 2022.
18. Bryan Ford. Identity and personhood in digital democracy: Evaluating inclusion, equality, security, and privacy in pseudonym parties and other proofs of personhood, 2020.
19. Bryan Ford and Jacob Strauss. An offline foundation for online accountable pseudonyms. In *Proceedings of the 1st Workshop on Social Network Systems*, SocialNets '08, page 31–36, New York, NY, USA, 2008. Association for Computing Machinery.
20. Jonathan Frankle, Sunoo Park, Daniel Shaar, Shafi Goldwasser, and Daniel Weitzner. Practical accountability of secret processes. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 657–674, 2018.
21. Matthew Franklin and Haibin Zhang. A framework for unique ring signatures. *Cryptology ePrint Archive*, 2012.

22. Matthew Franklin and Haibin Zhang. Unique ring signatures: A practical construction. In *Financial Cryptography and Data Security: 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers 17*, pages 162–170. Springer, 2013.
23. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. Cryptology ePrint Archive, Paper 2017/620, 2017, 2017. `https://eprint.iacr.org/2017/620`.
24. Eiichiro Fujisaki. Sub-linear size traceable ring signatures without random oracles. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, 95(1):151–166, 2012.
25. Eiichiro Fujisaki and Koutarou Suzuki. Traceable ring signature. In *International Workshop on Public Key Cryptography*, pages 181–200. Springer, 2007.
26. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953, 2019. `https://eprint.iacr.org/2019/953`.
27. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. Cryptology ePrint Archive, Paper 2012/215, 2012, 2012. `https://eprint.iacr.org/2012/215`.
28. Sharon Goldberg, Leonid Reyzin, Dimitrios Papadopoulos, and Jan Včelák. Verifiable Random Functions (VRFs). Internet-Draft draft-irtf-cfrg-vrf-10, Internet Engineering Task Force, Nov 2021. Work in Progress.
29. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. IACR ePrint Archive 2016/260.
30. Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *Journal of the ACM (JACM)*, 59(3):1–35, 2012.
31. Kobi Gurkan, Koh Wei Jie, and Barry Whitehat. Community proposal: Semaphore: Zero-knowledge signaling on ethereum. *ZKProof Standards*, 2020.
32. Joe Kilian. Uses of randomness in algorithms and protocols. PhD Thesis. Massachusetts Institute of Technology, 1990.
33. Stefan Köpsell, Rolf Wendolsky, and Hannes Federrath. Revocable anonymity. In *International Conference on Emerging Trends in Information and Communication Security*, pages 206–220. Springer, 2006.
34. Joseph K Liu, Man Ho Au, Willy Susilo, and Jianying Zhou. Linkable ring signature with unconditional anonymity. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):157–165, 2013.
35. Joseph K Liu and Duncan S Wong. Linkable ring signatures: Security models and new schemes. In *Computational Science and Its Applications–ICCSA 2005: International Conference, Singapore, May 9-12, 2005, Proceedings, Part II 5*, pages 614–623. Springer, 2005.
36. Giulio Malavolta and Dominique Schröder. Efficient ring signatures in the standard model. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II 23*, pages 128–157. Springer, 2017.
37. Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.
38. Lan Nguyen and Rei Safavi-Naini. Dynamic k-times anonymous authentication. In *Applied Cryptography and Network Security: Third International Conference, ACNS 2005, New York, NY, USA, June 7-10, 2005. Proceedings 3*, pages 318–333. Springer, 2005.
39. Tuong Ngoc Nguyen, Anh The Ta, Huy Quoc Le, Dung Hoang Duong, Willy Susilo, Fuchun Guo, Kazuhide Fukushima, and Shinsaku Kiyomoto. Efficient unique ring signatures from lattices. In *European Symposium on Research in Computer Security*, pages 447–466. Springer, 2022.
40. Dimitrios Papadopoulos, Duane Wessels, Shumon Huque, Moni Naor, Jan Včelák, Leonid Reyzin, and Sharon Goldberg. Making nsec5 practical for dnssec. Cryptology ePrint Archive, Paper 2017/099, 2017. `https://eprint.iacr.org/2017/099`.
41. Trevor Perrin. The xeddsa and vxeddsa signature schemes. Revision 1, 2016-10-20. `https://signal.org/docs/specifications/xeddsa/`.
42. Anh The Ta, Thanh Xuan Khuc, Tuong Ngoc Nguyen, Huy Quoc Le, Dung Hoang Duong, Willy Susilo, Kazuhide Fukushima, and Shinsaku Kiyomoto. Efficient unique ring signature for blockchain privacy protection. In *Information Security and Privacy: 26th Australasian Conference, ACISP 2021, Virtual Event, December 1–3, 2021, Proceedings 26*, pages 391–407. Springer, 2021.

43. Isamu Teranishi, Jun Furukawa, and Kazue Sako. K-times anonymous authentication. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 308–322. Springer, 2004.
44. Luis Von Ahn, Andrew Bortz, Nicholas J Hopper, and Kevin O'Neill. Selectively traceable anonymity. In *Privacy Enhancing Technologies: 6th International Workshop, PET 2006, Cambridge, UK, June 28-30, 2006, Revised Selected Papers 6*, pages 208–222. Springer, 2006.
45. Shuang Yao, Dawei Zhang, et al. An anonymous verifiable random function with applications in blockchain. *Wireless Communications and Mobile Computing*, 2022, 2022.

A Security of Our ring VRF Construction

A.1 Security of the First ring VRF Construction

We describe the simulation in Section 4. We show below the indistinguishability of our simulation.

The proof of Lemma 2 is below:

Proof. The evaluation outputs of the ring signatures in the ideal world identical to the real world protocol because the outputs are randomly selected by $\mathcal{F}_{\mathsf{rvrf}}$ as the random oracle H in the real protocol. The only difference is the ring signatures of honest parties (See Algorithm 1) since the pre-output W and π_1 are generated differently in Algorithm 1 than $\mathsf{rVRF.Sign}$. The distribution of $\pi_{eval} = (c, s_1, s_2)$ and compk generated by Algorithm 1 and the distribution of $\pi_{eval} = (c, s_1, s_2)$ and compk generated by $\mathsf{rVRF.Sign}$ are from uniform distribution so they are indistinguishable. So, we are left to show that the anonymous key W selected randomly from $\mathbb{G}$ and pre-output W generated by $\mathsf{rVRF.Sign}$ are indistinguishable given pk.

Case 1 ($\mathsf{pk} \leftarrow \mathsf{Com.Commit}(x, r)$): Since pk is a perfectly hiding commitment, then pk is uniformly random and independent from x. Therefore, the anonymous key W selected randomly from $\mathbb{G}$ and pre-output $W = xG$ generated by $\mathsf{rVRF.Sign}$ are indistinguishable given pk.

Case 2 ($\mathsf{pk} = xG$): In this case, pk is not independent from the secret key. Therefore, we need more to show the indistinguishability. We show this under the assumption that the DDH problem is hard. In other words, we show that if there exists a distinguisher $\mathcal{D}$ that distinguishes honest signatures in the ideal world and honest signatures in the real protocol then we construct another adversary $\mathcal{B}$ which breaks the DDH problem. We use the hybrid argument to show this. We define hybrid simulations H_i where the signatures of first i honest parties are computed as described in $\mathsf{rVRF.Sign}$ and the rest are computed as in $\mathcal{F}_{\mathsf{rvrf}}$. Without loss of generality, $\mathsf{P}_1, \mathsf{P}_2, \ldots, \mathsf{P}_{n_h}$ are the honest parties. Thus, H_0 is equivalent to the honest signatures generated in the ideal world and H_{n_h} is equivalent to honest signatures in the real world. We construct an adversary $\mathcal{B}$ that breaks the DDH problem given that there exists an adversary $\mathcal{D}$ that distinguishes hybrid games H_i and H_{i+1} for $0 \leq i < n_h$. $\mathcal{B}$ receives the DDH challenges $X, Y, Z \in \mathbb{G}$ from the DDH game and simulates the game against $\mathcal{D}$ as follows. $\mathcal{B}$ runs a simulated copy of $\mathcal{Z}$ and starts to simulate $\mathcal{F}_{\mathsf{rvrf}}$ and Sim for $\mathcal{Z}$. For this, it first runs the simulated copy of $\mathcal{A}$ as Sim does. $\mathcal{B}$ publishes $\mathbb{G}, G = Y, K$ as parameters of the ring VRF protocol. $\mathcal{B}$ generates the public key of all honest parties' key as usual by running $\mathsf{rVRF.KeyGen}$ as Sim does except party P_{i+1}. It lets the public key of P_{i+1} be X.

While simulating $\mathcal{F}_{\mathsf{rvrf}}$, $\mathcal{B}$ simulates the ring signatures of first i parties by running $\mathsf{rVRF.Sign}$ and the parties $\mathsf{P}_{i+2}, \ldots, \mathsf{P}_{n_h}$ by running Algorithm 1 where W is selected randomly. The simulation of P_{i+1} is different. Whenever P_{i+1} needs to sign an input in and message ad, it obtains $\mathsf{inbase} = H_{\mathbb{G}}(m) = hY$ from the oracle $\mathtt{H}_p$ and lets $W = hZ$. Then it sets $\mathsf{compk} = X + \mathsf{b}K$ and $\pi_{\mathtt{eval}} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{eval}}.\mathsf{Simulate}(\mathsf{compk}, W, \mathsf{in})$ and $\pi_{ring} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{\mathtt{ring}}}.\mathsf{Simulate}(\mathsf{compk}, \mathsf{comring})$ by inputting ad in the random oracle H. Remark that if (X, Y, Z) is a DH triple (i.e., $\mathsf{DH}(X, Y, Z) \rightarrow 1$), P_{i+1} is simulated as in in our construction because $W = \mathsf{sk}G$ in this case. Otherwise, P_{i+1} is simulated as in the ideal

$\mathcal{F}_{\mathsf{rvrf}}$ runs a PPT algorithms Gen_{sign} during the execution and is parametrized with sets $\mathcal{S}_{eval}$ and $\mathcal{S}_W$ where $\mathcal{S}_{eval}$ and $\mathcal{S}_W$ generated by a set up function $\mathsf{Setup}(1^\lambda)$.

[Key Generation.] upon receiving a message $(\mathsf{keygen}, \mathsf{sid})$ from P_i, send $(\mathsf{keygen}, \mathsf{sid}, \mathsf{P}_i)$ to the simulator Sim. Upon receiving a message $(\mathsf{verificationkey}, \mathsf{sid}, \mathsf{sk}, \mathsf{pk})$ from Sim, verify that sk or pk has not been recorded before for sid in `signing_keys`. If it is the case, store the value sk, pk in the table `signing_keys` under P_i and return $(\mathsf{verificationkey}, \mathsf{sid}, \mathsf{pk})$ to P_i.

[Honest Ring VRF Signature and Evaluation.] upon receiving a message $(\mathsf{sign}, \mathsf{sid}, \mathsf{ring}, \mathsf{pk}_i, \mathsf{ad}, \mathsf{in})$ from P_i, verify that $\mathsf{pk}_i \in \mathsf{ring}$ and that there exists a public key pk_i associated to P_i in `signing_keys`. If it is not the case, just ignore the request. If there exists no W' such that $\texttt{anonymous\_key\_map}[\mathsf{in}, W'] = \mathsf{pk}_i$, let $W \leftarrow\!\$\ \mathcal{S}_W$ and let $\mathsf{out} \leftarrow\!\$\ \mathcal{S}_{eval}$. Set $\texttt{anonymous\_key\_map}[\mathsf{in}, W] = \mathsf{pk}_i$ and set $\texttt{evaluations}[\mathsf{in}, W] = \mathsf{out}$. In any case (except ignoring), obtain W, out where $\texttt{anonymous\_key\_map}[\mathsf{in}, W] = \mathsf{pk}_i$, $\texttt{evaluations}[\mathsf{in}, W] = \mathsf{out}$ and $(\mathsf{sk}, \mathsf{pk})$ is in `signing_keys`. Then run $\mathsf{Gen}_{sign}(\mathsf{ring}, \mathsf{sk}, \mathsf{pk}, \mathsf{ad}, \mathsf{in}) \to \sigma$. Let $\sigma = (\sigma, W)$ and record $[\mathsf{in}, \mathsf{ad}, W, \mathsf{ring}, \sigma, 1]$. Return $(\mathsf{signature}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, \mathsf{in}, \mathsf{out}, \sigma)$ to P_i.

[Malicious Ring VRF Evaluation.] upon receiving a message $(\mathsf{eval}, \mathsf{sid}, \mathsf{pk}_i, W, \mathsf{in})$ from Sim, if pk_i is recorded under an honest party's identity or if there exists $W' \neq W$ where $\texttt{anonymous\_key\_map}[\mathsf{in}, W'] = \mathsf{pk}_i$, ignore the request. Otherwise, record in the table `signing_keys` the value $(\perp, \mathsf{pk}_i)$ under Sim if $(., \mathsf{pk}_i)$ is not in `signing_keys`. If $\texttt{anonymous\_key\_map}[\mathsf{in}, W]$ is not defined before, set $\texttt{anonymous\_key\_map}[\mathsf{in}, W] = \mathsf{pk}_i$ and let $\mathsf{out} \leftarrow\!\$\ \mathcal{S}_{eval}$ and set $\texttt{evaluations}[\mathsf{in}, W] = \mathsf{out}$. In any case (except ignoring), obtain $\mathsf{out} = \texttt{evaluations}[\mathsf{in}, W]$ and return $(\mathsf{evaluated}, \mathsf{sid}, \mathsf{in}, \mathsf{pk}_i, W, \mathsf{out})$ to P_i. **[Corruption:]** upon receiving $(\mathsf{corrupt}, \mathsf{sid}, \mathsf{P}_i)$ from Sim, remove (x_i, pk_i) from $\texttt{signing\_keys}[\mathsf{P}_i]$ and store them to `signing_keys` under Sim. Return $(\mathsf{corrupted}, \mathsf{sid}, \mathsf{P}_i)$.

[Malicious Requests of Signatures.] upon receiving a message $(\mathsf{signs}, \mathsf{sid}, W, \mathsf{ad}, \mathsf{in})$ from Sim, obtain all existing valid signatures σ such that $[\mathsf{in}, \mathsf{ad}, W, ., \sigma, 1]$ is recorded and add them in a list $\mathcal{L}_\sigma$. Return $(\mathsf{signs}, \mathsf{sid}, W, \mathsf{ad}, \mathsf{in}, \mathcal{L}_\sigma)$ to Sim.

[Ring VRF Verification.] upon receiving a message $(\mathsf{verify}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, \mathsf{in}, \sigma)$ from a party, do the following:

C1 If there exits a record $[\mathsf{in}, \mathsf{ad}, W, \mathsf{ring}, \sigma, b']$, set $b = b'$.

C2 Else if $\texttt{anonymous\_key\_map}[\mathsf{in}, W]$ is an honest verification key and there exists a record $[\mathsf{in}, \mathsf{ad}, W, \mathsf{ring}, \sigma', 1]$ for any σ', then let $b = 1$ and record $[\mathsf{in}, \mathsf{ad}, W, \mathsf{ring}, \sigma, 1]$.

C3 Else relay the message $(\mathsf{verify}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, \mathsf{in}, \sigma)$ to Sim and receive back the message $(\mathsf{verified}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, \mathsf{in}, \sigma, b_{\mathsf{Sim}}, \mathsf{pk}_{\mathsf{Sim}})$. Then check the following:

1. If $\mathsf{pk}_{\mathsf{Sim}}$ is an honest verification key, set $b = 0$.
2. Else if $W \notin \mathcal{W}[\mathsf{in}, \mathsf{ring}]$ and $|\mathcal{W}[\mathsf{in}, \mathsf{ring}]| \geq |\mathsf{ring}_{mal}|$ where ring_{mal} is a set of malicious keys in ring, set $b = 0$. .
3. Else if there exists $W' \neq W$ where $\texttt{anonymous\_key\_map}[\mathsf{in}, W'] = \mathsf{pk}_{\mathsf{Sim}}$, set $b = 0$.
4. Else set $b = b_{\mathsf{Sim}}$.

In the end, record $[\mathsf{in}, \mathsf{ad}, W, \mathsf{ring}, \sigma, 0]$ if it is not stored. If $b = 0$, let $\mathsf{out} = \perp$. Otherwise, do the following:

– if $W \notin \mathcal{W}[\mathsf{in}, \mathsf{ring}]$, add W to $\mathcal{W}[\mathsf{in}, \mathsf{ring}]$.
– if $\texttt{evaluations}[\mathsf{in}, W]$ is not defined, sample $y \leftarrow\!\$\ \mathcal{S}_{eval}$. Then, set $\texttt{anonymous\_key\_map}[\mathsf{in}, W] = \mathsf{pk}_{\mathsf{Sim}}$ and $\texttt{evaluations}[\mathsf{in}, W] = \mathsf{out}$.
– otherwise, set $\mathsf{out} = \texttt{evaluations}[\mathsf{in}, W]$.

Finally, output $(\mathsf{verified}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, \mathsf{in}, \sigma, \mathsf{out}, b)$ to the party.

[Corruption:] upon receiving $(\mathsf{corrupt}, \mathsf{sid}, \mathsf{P}_i)$ from Sim, remove $(\mathsf{sk}_i, \mathsf{pk}_i)$ from $\texttt{signing\_keys}[\mathsf{P}_i]$ and store them to `signing_keys` under Sim. Return $(\mathsf{corrupted}, \mathsf{sid}, \mathsf{P}_i)$.

Fig. 4. Functionality $\mathcal{F}_{\mathsf{rvrf}}$.

world because W is random. So, if $\mathsf{DH}(X, Y, Z) \to 1$, Sim simulates H_{i+1}. Otherwise, it simulates H_i. In the end of the simulation, if $\mathcal{D}$ outputs i, Sim outputs 0 meaning $\mathsf{DH}(X, Y, Z) \to 0$. Otherwise, it outputs $i+1$. The success probability of Sim is equal to the success probability of $\mathcal{D}$ which distinguishes H_i and H_{i+1}. Since DDH problem is hard, Sim has negligible advantage in the DDH game. So, $\mathcal{D}$ has a negligible advantage too. Hence, from the hybrid argument, we can conclude that H_0 which corresponds the output of honest parties in the ring VRF protocol and H_q which corresponds to the output of honest parties in ideal world are indistinguishable.

This concludes the proof of showing the output of honest parties in the ideal world are indistinguishable from the output of the honest parties in the real protocol.

Next we show that the simulation executed by Sim against $\mathcal{A}$ is indistinguishable from the real protocol execution.

The proof of Lemma 3 is below:

Proof. The simulation against the real world adversary $\mathcal{A}$ is identical to the real protocol except the cases where Sim aborts. Abort-1 cannot happen as we explained during the simulation. Abort-2 happens if Gen_{sign} generates the same compk for two different signatures. This happens if $\mathcal{F}_{\mathsf{rvrf}}$ select same compk for two different honest signature which happens with a negligible probability. Now, we are left with the abort case (Abort-3) during the verification. For this, we show that if there exists an adversary $\mathcal{A}$ which makes Sim abort during the simulation, then we construct another adversary $\mathcal{B}$ which breaks either the CDH problem or the binding property of Com.

Consider a CDH game in a prime p-order group $\mathbb{G}$ with the challenges $G, U, V \in \mathbb{G}$. The CDH challenges are given to the simulator $\mathcal{B}$. Then $\mathcal{B}$ runs a simulated copy of $\mathcal{Z}$ and starts to simulate $\mathcal{F}_{\mathsf{rvrf}}$ and Sim for $\mathcal{Z}$. For this, it first runs the simulated copy of $\mathcal{A}$ as Sim does. $\mathcal{B}$ provides $(\mathbb{G}, p, G, K)$ as a public parameter of the ring VRF protocol to $\mathcal{A}$.

Whenever $\mathcal{B}$ needs to generate a ring signature of input in and message ad on behalf of an honest party, it behaves exactly as $\mathcal{F}_{\mathsf{rvrf}}$ except that it runs Algorithm 3 to generate the signature.

Algorithm 3 $\mathsf{Gen}_{sign}(\mathsf{ring}, W, \mathsf{pk}, \mathsf{ad}, \mathsf{in})$

```
1: $\mathsf{compk} \leftarrow\!\$\ \mathbb{G}$
2: $\pi_{\mathtt{eval}} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{eval}}.\mathsf{Simulate}(\mathsf{compk}, W, \mathsf{in})$
3: $\mathsf{comring}, \mathsf{opring} \leftarrow \mathsf{rVRF.CommitRing}(\mathsf{ring})$
4: $\pi_{\mathtt{ring}} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{\mathtt{ring}}}.\mathsf{Simulate}(\mathsf{comring}, \mathsf{compk})$
5: return $\sigma = (\mathsf{compk}, \pi_{\mathtt{ring}}, \mathsf{comring}, \pi_{\mathtt{eval}})$
```

Clearly the ring signature of an honest party outputted by Sim (remember $\mathcal{F}_{\mathsf{rvrf}}$ generates it by Algorithm 1) and the ring signature generated by $\mathcal{B}$ are indistinguishable. Remark that $\mathcal{B}$ does not need to set H_p any more as Sim so that $\pi_{\mathtt{eval}}$ verifies because Gen_{sign} in Algorithm 3 does it while simulating the proof for $\mathcal{R}_{eval}$. Therefore, the simulation of H_p is simulated as a usual random oracle by $\mathcal{B}$.

In order to generate the public keys of honest parties, $\mathcal{B}$ picks a random $r_x \in \mathbb{F}_p$ and sets $X = r_x V$. If $\mathsf{rVRF.KeyGen}$ generates a public key as $\mathsf{pk} = \mathsf{sk}G$, it lets pk be X otherwise it picks a random public key pk . Remark that $\mathcal{B}$ never needs to know the secret key of honest parties to simulate them since $\mathcal{B}$ selects anonymous keys randomly and generates the ring signatures without the secret keys. Since the public key generated by $\mathsf{rVRF.KeyGen}$ is random and independent from the secret key, $\mathcal{B}$'s key generation is indistinguishable from $\mathsf{rVRF.KeyGen}$, if $\mathsf{rVRF.KeyGen}$ generates a public key as a commitment.

$\mathcal{B}$ simulates $\mathcal{F}_{\mathsf{rvrf}}$ as described but with the following difference: whenever $\mathcal{F}_{\mathsf{rvrf}}$ sets up $\mathtt{evaluations}[\mathsf{in}, W]$ it queries in, W to the random oracle H. $\mathcal{B}$ simulates the random oracle H as a usual random oracle. The only difference from the simulation of H by Sim is that $\mathcal{B}$ does not ask for the output of $H(\mathsf{in}, W)$ to $\mathcal{F}_{\mathsf{rvrf}}$ but it does not make any difference because now $\mathcal{F}_{\mathsf{rvrf}}$ asks for it. $\mathcal{B}$ also simulates H_{ring} for the ring commitments as a usual random oracle. Simulation of $H_{\mathbb{G}}$ by $\mathcal{B}$ returns hU instead of hG.

During the simulation, when $\mathcal{A}$ outputs a signature $\sigma = (\mathsf{compk}, \pi_{ring}, \pi_{eval}, \mathsf{comring}, W)$ of an input in and message ad which is not recorded in $\mathcal{B}$'s record as $\mathcal{F}_{\mathsf{rvrf}}$ has, $\mathcal{B}$ runs $\mathsf{rVRF.Verify}(\mathsf{comring}, \mathsf{in}, \mathsf{ad}, \sigma)$. If it verifies, it runs the extractor algorithm of $\mathsf{NIZK}_{\mathcal{R}_{\mathtt{ring}}}$ and obtains $b, \mathsf{opring}, \mathsf{sk}$ in version 1 and obtains b, opring in version 2. In both cases, the simulation is the same because $\mathcal{B}$ does not need sk. Since $\mathsf{opring} = (\mathsf{pk}, \pi_{\mathsf{comring}})$ contains a valid proof, it obtains ring by running the extractor algorithm of $\mathsf{NARK}_{\mathcal{R}_{\mathsf{comring}}}$. Then, it computes $X = \mathsf{compk} - \mathsf{b}\,K$. If pk is not an honest key then $\mathcal{B}$ adds W to $\mathcal{W}[m, \mathsf{ring}]$. Then, it runs the extractor algorithm of $\mathsf{NIZK}_{\mathcal{R}_{eval}}$ and obtains $(\hat{x}, \hat{\mathsf{b}})$ such that $\mathsf{compk} = \hat{x}G + \hat{\mathsf{b}}\,K$ and $W = \hat{x}H_{\mathbb{G}}(m)$. If $W \notin \mathcal{W}[\mathsf{in}, \mathsf{ring}]$, $\mathcal{B}$ increments $\mathsf{counter}[\mathsf{in}, \mathsf{ring}]$ and adds W to $\mathcal{W}[\mathsf{in}, \mathsf{ring}]$.

If X is generated by $\mathcal{B}$ during a key generation process of an honest party and $X = \hat{x}G$, $\mathcal{B}$ solves the CDH problem as follows: $W = \hat{x}hU$ where $h = \mathtt{H_p}[m]$. Since $X = rV$, $W = xhuG = rhuV$. So, $\mathcal{B}$ outputs $r^{-1}h^{-1}W$ as a CDH solution and simulation ends. Remark that this case happens when Sim aborts because of 1.

If $\mathcal{W}[m, \mathsf{ring}] = t' > |\mathsf{ring}_{mal}| = t$, $\mathcal{B}$ obtains all the signatures $\{\sigma_i\}_{i=1}^{t'}$ that make $\mathcal{B}$ to add an anonymous key to $\mathcal{W}[m, \mathsf{ring}]$. Then it solves the CDH problem as follows: Remark that this case happens when Sim aborts because of C2.

For all $\sigma_j = (\mathsf{compk}_j, \pi_{ring_j}, \mathsf{comring}, \pi_{eval_j}, W_j) \in \{\sigma_i\}_{i=1}^{t'}$, $\mathcal{B}$ runs the extractor for $\mathcal{R}_{\mathtt{ring}}$ and obtains $\mathsf{opring}_j, \mathsf{b}_j, \mathsf{sk}_j$ in version 1 and $(\mathsf{opring}_j, \mathsf{b}_j)$ in version 2. Then it obtains the public key $\mathsf{pk}_j \in \mathsf{opring}_j$ where $\mathsf{pk}_j \in \mathsf{ring}$ and $X_j = \mathsf{compk} - \mathsf{b}_j K = x_j G$.Then, it adds X_j to a list $\mathcal{X}$ and pk_j to a set $\mathcal{PK}$. One of the following cases happens:

1. All X_j in $\mathcal{X}$ are different and $|\mathcal{PK}| \leq t$: This only happens if we are in version 1. Because in version 2, $\mathsf{pk} = x_j G = X_j$. In version 1, each $\mathsf{pk}_j \in \mathcal{PK}$ commits to a secret key x_j such that $x_j = \mathsf{Com.Open}(\mathsf{pk}_j, r_j)$. If all X_j's are different and $|\mathcal{PK}| \leq t$, then there exists a $\mathsf{pk}_j \in \mathcal{PK}$ where $x_j = \mathsf{Com.Open}(\mathsf{pk}_j, r_j)$ and $x'_j = \mathsf{Com.Open}(\mathsf{pk}_j, r'_j)$ such that $x_j G, x'_j G \in \mathcal{X}$. So, it means that the binding property of Com is broken which happens with a negligible probability. Therefore, $\mathcal{B}$ aborts with a negligible probability.
2. All X_j in $\mathcal{X}$ are different and $|\mathcal{PK}| > t$: If $\mathcal{B}$ is in this case, it means that there exists $X_a \in \mathcal{X}$ which belongs to an honest party because $\mathcal{PK}$ includes more keys than the malicious keys. This cannot happen at this point because $\mathcal{B}$ solves the CDH when $\mathcal{A}$ outputted σ_a when this happens as described above.
3. There exist at least two $X_a, X_b \in \mathcal{X}$ where $X_a = X_b$: $\mathcal{B}$ runs the extractor algorithm of $\mathsf{NIZK}_{\mathcal{R}_{eval}}$ for π_{ring_a} and π_{ring_b} and obtains $(\hat{x}_a, \hat{\mathsf{b}}_a)$ and $(\hat{x}_b, \hat{\mathsf{b}}_b)$, respectively such that $\mathsf{compk}_a = \hat{x}_a G + \hat{\mathsf{b}}_a\,K \mathsf{compk}_b = \hat{x}_b G + \hat{\mathsf{b}}_b\,K$ and $W_a = \hat{x}_a H_{\mathbb{G}}(m), W_b = \hat{x}_b H_{\mathbb{G}}(m)$. Since $W_a \neq W_b$, $\hat{x}_a \neq \hat{x}_b$. So, $\mathcal{B}$ can obtain two different and non trivial representation of $X_a = X_b$ i.e., $X_a = X_b = \hat{x}_a G + (\hat{\mathsf{b}}_a - \mathsf{b}_a)\,K = \hat{x}_b G + (\hat{\mathsf{b}}_b - \mathsf{b}_b)\,K$. Thus, $\mathcal{B}$ finds the discrete logarithm of $K = U$ in base G which is $u = \frac{\hat{x}_a - \hat{x}_b}{\hat{\mathsf{b}}_a - \mathsf{b}_a - \hat{\mathsf{b}}_b + \mathsf{b}_b}$. $\mathcal{B}$ outputs uV as a CDH solution.

So, the probability of $\mathcal{B}$ solves the CDH problem is equal to the probability of $\mathcal{A}$ breaks the forgery or uniqueness in the real protocol. Therefore, if there exists $\mathcal{A}$ that makes Sim aborts during the verification, then we can construct an adversary $\mathcal{B}$ that solves the CDH problem and breaking the binding property of Com except with a negligible probability.

This completes the security proof of our ring VRF protocol. □

A.2 Security of Our Protocol with $\mathsf{SpecialG}$

Lemma 4. *$\mathcal{F}_{\mathsf{rvrf}}$ running Algorithm 2 satisfies anonymity defined in Definition 7 assuming that* ZKCont *is a zero-knowledge as defined in Definition 8.*

Proof. We simulate $\mathcal{F}_{\mathsf{rvrf}}$ with Algorithm 1 against $\mathcal{D}$. Assume that the advantage of $\mathcal{D}$ is ϵ. Now, we reduce the anonymity game to the following game where we change the simulation of $\mathcal{F}_{\mathsf{rvrf}}$ by changing the Algorithm 1. In our change, we replace Line 2 and 3 of Algorithm 2 with $\mathsf{ZKCont.Simulate}(\mathsf{td}, \mathsf{comring}, \mathcal{R}^{\mathtt{inner}}_{\mathtt{ring}})$. Since ZKCont is zero knowledge, there exists an algorithm $\mathsf{ZKCont.Simulate}$ which generates a proof which is indistinguishable from the original proof and compk. Therefore, our reduced game is indistinguishable from the anonymity game. Since in this game, no key is used while generating the proof and W and compk is perfectly hiding, the probability that $\mathcal{D}$ wins the game is $\frac{1}{2}$. This means that ϵ is negligible.

We construct the same Sim described in the proof of Theorem 1 because it does not deploy any extractor or simulator of NIZK for $\mathcal{R}_{eval}$ and $\mathcal{R}_{\mathtt{ring}}$. Similarly, Lemma 2 applies here. The only difference is in Lemma 3 since Gen_{sign} is different than Algorithm 1. We first replace Gen_{sign} run by $\mathcal{B}$ in Algorithm 3 defined for Lemma 3 with Algorithm 4.

Algorithm 4 $\mathsf{Gen}_{sign}(\mathsf{ring}, W, \mathsf{pk}, \mathsf{ad}, m)$

```
1: $\mathsf{comring}, \mathsf{opring} \leftarrow \mathsf{rVRF.CommitRing}(\mathsf{ring})$
2: $\pi_{\mathtt{ring}}, \mathsf{compk} \leftarrow \mathsf{ZKCont.Simulate}(\mathsf{td}, \mathsf{comring})$
3: $\pi_{\mathtt{eval}} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{eval}}.\mathsf{Simulate}(\mathsf{compk}, W, m)$
4: **return** $\sigma = (\mathsf{compk}, \pi_{\mathtt{inner}}, \mathsf{comring}, \pi_{\mathtt{eval}})$
```

The other change is that we replace all extractors in Lemma 3 for $\mathcal{R}_{\mathtt{ring}}, \mathcal{R}_{eval}$ with the extractor for $\mathsf{NIZK}_{\mathcal{R}_{\mathtt{rvrf}}}$. $\mathcal{B}$ here is simpler than $\mathcal{B}$ in Lemma 3 because the secret key is the part of $\mathcal{R}_{\mathtt{rvrf}}$ while the secret key is not part of the witness in $\mathcal{R}_{\mathtt{ring}}$ for the case pk is defined as $\mathsf{sk}G$ (Version 2). When $\mathcal{B}$ sees a signature $\sigma = (\mathsf{compk}, \pi_{\mathtt{inner}}, \mathsf{comring}, \pi_{\mathtt{eval}})$ of in, it runs the extractor for $\mathsf{NIZK}_{\mathcal{R}_{\mathtt{rvrf}}}$ and obtains x, r, opring. Then, it lets X be xG. If X is generated for an honest party, it solves the CDH as described in Lemma 3 for the same case. If $\mathcal{W}[\mathsf{in}, \mathsf{ring}] = t' > |\mathsf{ring}_{mal}| = t$, it runs the extractors for $\mathsf{NIZK}_{\mathcal{R}_{\mathtt{ring}}}$ of all malicious signatures of in for ring and obtains $\{(x_j, r_j, \mathsf{opring}_j)\}_{j=1}^{t'}$. Then, for all $j \in [1, t']$, it adds $X_j = x_jG$ to $\mathcal{X}$ and $\mathsf{pk}_j = \mathsf{rVRF.OpenRing}(\mathsf{comring}, \mathsf{opring})$ to a list $\mathcal{PK}$. Then, the first two cases in Lemma 3 happens and $\mathcal{B}$ behaves the same. We note that here all sk_j's are different because preout_j's are different. Therefore, the last case in Lemma 3 does not happen.

B $\mathsf{NIZK}_{\mathcal{R}}$'s Security

Theorem 5. *If* $\mathsf{ZKCont}_{\mathcal{R}_1}$ *is a zk continuation for* $\mathcal{R}_1$ *and* $\mathsf{NIZK}_{\mathcal{R}'_2}$ *is a NIZK for* $\mathcal{R}'_2$ *for some appropriately chosen public parameters* pp*, then the* $\mathsf{NIZK}_{\mathcal{R}}$ *construction described above is a NIZK for* $\mathcal{R}$.

Proof. Putting together the results of Lemma 5, Lemma 6, Lemma 7 and we obtain the above statement.

Lemma 5 (Knowledge-soundness for $\mathsf{NIZK}_{\mathcal{R}}$). *If* $\mathsf{ZKCont}_{\mathcal{R}_1}$ *is a zk continuation for* $\mathcal{R}_1$ *and* $\mathsf{NIZK}_{\mathcal{R}'_2}$ *is a NIZK for* $\mathcal{R}'_2$ *for some appropriately chosen public parameters* pp*, then the* $\mathsf{NIZK}_{\mathcal{R}}$ *construction described above has knowledge-soundness for* $\mathcal{R}$.

Proof. This is easy to infer by linking together the extractors guaranteed for $\mathsf{ZKCont}_{\mathcal{R}_1}$ and $\mathsf{NIZK}_{\mathcal{R}_2'}$ due to their respective knowledge-soundness.

Next, we define *Special Perfect Completeness* for all λ, for every efficient adversary $\mathcal{A}$, for every $(\bar{z}, \bar{x}; \bar{w}_2) \in \mathcal{R}_2$ it holds

$$\begin{aligned}\Pr(&\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Verify}(\mathsf{crs}, \bar{y}, X', \pi_1', \mathcal{R}_1) = 1 \ \wedge\ \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{VerCom}(\mathsf{pp}, X', \bar{x}, b') = 1)) \\ &\qquad\qquad \Rightarrow\ \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Verify}(\mathsf{crs}_{\mathcal{R}}, X, \bar{z}, \pi_2) = 1 \mid \\ &(\mathsf{crs}, \mathsf{td}, \mathsf{pp}) \leftarrow \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Setup}(1^\lambda, \mathcal{R}_1), (crs_{\mathcal{R}_2'}, \mathsf{td}_{\mathcal{R}_2'}) \leftarrow \mathsf{NIZK}_{\mathcal{R}_2'}.\mathsf{Setup}(1^\lambda), \\ &(\bar{y}, X', \pi_1', b') \leftarrow A(\mathsf{crs}, \mathcal{R}_1), (X, \pi_1, b) \leftarrow \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Reprove}(\mathsf{crs}, X', \pi_1', b', \mathcal{R}_1), \\ &\pi_2 \leftarrow \mathsf{NIZK}_{\mathcal{R}_2'}.\mathsf{Prove}(crs_{\mathcal{R}_2'}, X, \bar{z}, \bar{x}, b, \bar{w}_2)) = 1\end{aligned}$$

Lemma 6 (Special Perfect Completeness). *If* $\mathsf{ZKCont}_{\mathcal{R}_1}$ *is a zk continuation for* $\mathcal{R}_1$ *and* $\mathsf{NIZK}_{\mathcal{R}_2'}$ *is a NIZK for* $\mathcal{R}_2'$ *for some appropriately chosen public parameters* pp*, then the* $\mathsf{NIZK}_{\mathcal{R}}$ *construction described above has special perfect completeness.*

Proof. This is easy to infer by combining the perfect completeness properties of $\mathsf{NIZK}_{\mathcal{R}_2'}$ axnd perfect completeness for $\mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Reprove}$.

Finally, we define

Zero-knowledge after Reproving a $\mathsf{ZKCont}_{\mathcal{R}_1}$ Proof For all $\lambda \in \mathbb{N}$, for every benign auxiliary input aux, for all $\bar{y}, \bar{x}, \bar{z}, \bar{w}_1, \bar{w}_2$ with $(\bar{y}, \bar{x}; \bar{w}_1) \in \mathcal{R}_1$ and $(\bar{z}, \bar{x}; \bar{w}_2) \in \mathcal{R}_2$, for all X', π_1', b', for every adversary A it holds:

$$\begin{aligned}&|\Pr(A(\mathsf{crs}, aux, \pi_1, \pi_2, X, \mathcal{R}) = 1 \mid (\mathsf{crs}, \mathsf{td}, \mathsf{pp}) \leftarrow \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Setup}(1^\lambda, \mathcal{R}_1), \\ &\quad (\pi_1, X, \_) \leftarrow \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Reprove}(\mathsf{crs}, X', \pi_1', b', \mathcal{R}_1), \pi_2 \leftarrow \mathsf{NIZK}_{\mathcal{R}_2'}.\mathsf{Prove}(crs_{\mathcal{R}_2'}, X, \bar{z}, \bar{x}, b, \bar{w}_2), \\ &\quad \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Verify}(\mathsf{crs}, \bar{y}, X', \pi_1', \mathcal{R}_1) = 1, \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{VerCom}(\mathsf{pp}, X', \bar{x}', b') = 1) \\ &- \Pr(A(\mathsf{crs}, aux, \pi_1, \pi_2, X, \mathcal{R}) = 1 \mid (\mathsf{crs}, \mathsf{td}, \mathsf{pp}) \leftarrow \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Setup}(1^\lambda, \mathcal{R}_1), \\ &\quad (\pi_1, \pi_2, X) \leftarrow \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Simulate}(\mathsf{td}, \bar{y}, \mathcal{R}_1), \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{Verify}(\mathsf{crs}, \bar{y}, X', \pi_1', \mathcal{R}_1) = 1, \\ &\quad \mathsf{ZKCont}_{\mathcal{R}_1}.\mathsf{VerCom}(\mathsf{pp}, X', \bar{x}', b') = 1)| \leq \mathsf{negl}(\lambda)\end{aligned}$$

Lemma 7 (ZK after Reproving a $\mathsf{ZKCont}_{\mathcal{R}_1}$ Proof). *If* $\mathsf{ZKCont}_{\mathcal{R}_1}$ *is a zk continuation for* $\mathcal{R}_1$ *and* $\mathsf{NIZK}_{\mathcal{R}_2'}$ *is a NIZK for* $\mathcal{R}_2'$ *for some appropriately chosen public parameters* pp*, then the* $\mathsf{NIZK}_{\mathcal{R}}$ *construction described above has zero-knowledge after reproving a* $\mathsf{ZKCont}_{\mathcal{R}_1}$ *proof.*

Proof. The statement follows from the perfect zero-knowledge w.r.t. $\mathcal{R}_1$ for $\mathsf{ZKCont}_{\mathcal{R}_1}$ and the zero-knowledge property of $\mathsf{NIZK}_{\mathcal{R}_2'}$ w.r.t. $\mathcal{R}_2'$.

C Ring VRF Variations

In this section, we give a ring VRF functionality which gives more security properties than the basic ring VRF functionality $\mathcal{F}_{\mathsf{rvrf}}$ that we define in Figure 4.

C.1 Secret Ring VRF

We also define another version of $\mathcal{F}_{\mathsf{rvrf}}$ that we call $\mathcal{F}^s_{\mathsf{rvrf}}$. $\mathcal{F}^s_{\mathsf{rvrf}}$ operates as $\mathcal{F}_{\mathsf{rvrf}}$. In addition, it also lets a party generate a secret element to check whether it satisfies a certain relation i.e., $((m, y), (\eta, \mathsf{pk}_i)) \in \mathcal{R}$ where η is the secret random element. If it satisfies the relation, then $\mathcal{F}^s_{\mathsf{rvrf}}$ generates a proof. Proving works as $\mathcal{F}_{zk}$ [30] except that a part of the witness (η) is generated randomly by the functionality. $\mathcal{F}^s_{\mathsf{rvrf}}$ is useful in applications where a party wants to show that the random output y satisfies a certain relation without revealing his identity.

$\mathcal{F}^s_{\mathsf{rvrf}}$ for a relation $\mathcal{R}$ behaves exactly as $\mathcal{F}_{\mathsf{rvrf}}$. Differently, it has an algorithm Gen_π and it additionally does the following:

Secret Element Generation of Malicious Parties. upon receiving a message $(\mathsf{secret\_rand}, \mathsf{sid}, \mathsf{ring}, \mathsf{pk}, W, m)$ from Sim, verify that $\mathtt{anonymous\_key\_map}[m, W] = \mathsf{pk}_i$. If that was not the case, just ignore the request. If $\mathtt{secrets}[m, W]$ is not defined, obtain $y = \mathtt{evaluations}[m, W]$. Then, run $\mathsf{Gen}_\eta(m, \mathsf{pk}_i, y) \to \eta$ and store $\mathtt{secrets}[m, W] = \eta$. Obtain $\eta = \mathtt{secrets}[m, W]$ and return $(\mathsf{secret\_rand}, \mathsf{sid}, \mathsf{ring}, W, \eta)$ to P_i.

Secret Random Element Proof. upon receiving a message $(\mathsf{secret\_rand}, \mathsf{sid}, \mathsf{pk}, W, m)$ from P_i, verify that $\mathtt{anonymous\_key\_map}[m, W] = \mathsf{pk}_i$. If that was not the case, just ignore the request. If $\mathtt{secrets}[m, W]$ is not defined, run $\mathsf{Gen}_\eta(m, \mathsf{pk}_i, y) \to \eta$ and store $\mathtt{secrets}[m, W] = \eta$. Obtain $\eta \leftarrow \mathtt{secrets}[m, W]$ and $y \leftarrow \mathtt{evaluations}[m, W]$. If $((m, y), (\eta, \mathsf{pk}_i)) \in \mathcal{R}$, run $\mathsf{Gen}_\pi(W, m) \to \pi$ and add π to a list $\mathtt{zkproofs}[m, W]$. Else, let π be $\perp$. Return $(\mathsf{secret\_rand}, \mathsf{sid}, W, \eta, \pi)$ to P_i.

Secret Verification. upon receiving a message $(\mathsf{secret\_verify}, \mathsf{sid}, W, m, \pi)$, relay the message to Sim and receive $(\mathsf{secret\_verify}, \mathsf{sid}, W, m, \pi, \mathsf{pk}, \eta)$. Then,

- if $\pi \in \mathtt{zkproofs}[m, W, \mathsf{ring}]$, set $b = 1$.
- else if $\mathtt{secrets}[W, m] = \eta$ and $((m, y, \mathsf{ring}), (\eta, \mathsf{pk}_i)) \in \mathcal{R}$, set $b = 1$ and add to the list $\mathtt{zkproofs}[m, W, \mathsf{ring}]$.
- else set $b = 0$.

Send $(\mathsf{verification}, \mathsf{sid}, \mathsf{ring}, W, m, \pi, b)$ to P_i.

Fig. 5. Functionality $\mathcal{F}^s_{\mathsf{rvrf}}$.

D SpecialG as Instantiation of ZKCont

Below we describe SpecialG in more details. We start by giving a reminder about Quadratic Arithmetic Program (QAP) [10], [27] and related $\mathcal{R}_\mathcal{Q}$ in a standard way.

Definition 9 (QAP). *A Quadratic Arithmetic Program (QAP) $\mathcal{Q} = (\mathcal{A}, \mathcal{B}, \mathcal{C}, t(X))$ of size m and degree d over a finite field $\mathbb{F}_q$ is defined by three sets of polynomials $\mathcal{A} = \{a_i(X)\}_{i=0}^m$, $\mathcal{B} = \{b_i(X)\}_{i=0}^m$, $\mathcal{C} = \{c_i(X)\}_{i=0}^m$, each of degree less than $d-1$ and a target degree d polynomial $t(X)$. Given $\mathcal{Q}$ we define $\mathcal{R}_\mathcal{Q}$ as the set of pairs $((\bar{y}, \bar{x}); \bar{w}) \in \mathbb{F}_q^l \times \mathbb{F}_q^{n-l} \times \mathbb{F}_q^{m-n}$ for which it holds that there exist a polynomial $h(X)$ of degree at most $d-2$ such that:*

$$(\sum_{k=0}^m v_k \cdot a_k(X)) \cdot (\sum_{k=0}^m v_k \cdot b_k(X)) = (\sum_{k=0}^m v_k \cdot c_k(X)) + h(X)t(X) \quad (*)$$

where $\bar{v} = (v_0, \ldots, v_m) = (1, x_1, \ldots, x_n, w_1, \ldots w_{m-n})$ and $\bar{y} = (x_1, \ldots, x_l)$ and $\bar{x} = (x_{l+1}, \ldots, x_n)$ and $\bar{w} = (w_1, \ldots, w_{m-n})$.

Given notation provided in section 2, in particular elliptic curve $\mathbf{G}$, its pairing e and the related source, target groups and generators, we introduce

Definition 10 (Specialised Groth16 (SpecialG)). *Let $\mathcal{R}_\mathcal{Q}$ be as mentioned above. We call specialised Groth16 for relation $\mathcal{R}_\mathcal{Q}$ the following: instantiation of the zero-knowledge continuation notion from Definition 8:*

- $\mathsf{SpecialG.Setup} : (1^\lambda, \mathcal{R}_\mathcal{Q}) \mapsto (\mathsf{crs}, \mathsf{td}, \mathsf{pp})$.
  Let $\alpha, \beta, \gamma, \delta, \tau, \eta \xleftarrow{\$} \mathbb{F}_q^*$. Let $\mathsf{td} = (\alpha, \beta, \gamma, \delta, \tau, \eta)$.

Let $\mathsf{crs} = (\bar{\sigma}_1, \bar{\sigma}_2)$ *where*

$$\begin{aligned}\bar{\sigma}_1 = (&\alpha \cdot G_1, \beta \cdot G_1, \delta \cdot G_1, \{\tau_i \cdot G_1\}_{i=0}^{d-1},\\ &\left\{\frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\gamma} \cdot G_1\right\}_{i=1}^{n}, \frac{\eta}{\gamma} \cdot G_1,\\ &\left\{\frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\delta} \cdot G_1\right\}_{i=n+1}^{m},\\ &\left\{\frac{1}{\delta}\sigma^i t(\sigma) \cdot G_1\right\}_{i=0}^{d-2}, \frac{\eta}{\delta} \cdot G_1),\\ \bar{\sigma}_2 = (&\beta \cdot G_2, \gamma \cdot G_2, \delta \cdot G_2, \{\tau^i \cdot G_2\}_{i=0}^{d-1}).\end{aligned}$$

$\mathsf{pp} = \left(\left\{\frac{\beta a_i(\tau)+\alpha b_i(\tau)+c_i(\tau)}{\gamma} \cdot G_1\right\}_{i=l+1}^{n}, \frac{\eta}{\gamma} \cdot G_1\right)$.
Moreover, for simplicity and later use, we call
$K_\gamma = \frac{\eta}{\gamma} \cdot G_1$ *and* $K_\delta = \frac{\eta}{\delta} \cdot G_1$.

– $\mathsf{SpecialG.Preprove} : (\mathsf{crs}, \bar{y}, \bar{x}, \bar{w}_1, \mathcal{R}_\mathcal{Q}) \mapsto (X', \pi', b')$ *such that*

$$\begin{aligned} &b' = 0; r, s \xleftarrow{\$} \mathbb{F}_p; X' = \sum_{i=l+1}^{n} v_i \cdot \frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\gamma} \cdot G_1;\\ &o = \alpha + \sum_{i=0}^{m} v_i \cdot a_i(\tau) + r \cdot \delta; u = \beta + \sum_{i=0}^{m} v_i \cdot b_i(\tau) + s \cdot \delta;\\ &v = \frac{\sum_{i=n+1}^{m}(v_i(\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau))) + h(\tau)t(\tau)}{\delta} +\\ &\quad + o \cdot s + u \cdot r - r \cdot s \cdot \delta;\\ &\pi' = (o \cdot G_1, u \cdot G_2, v \cdot G_1),\end{aligned}$$

where $\bar{y} = (x_1, \ldots, x_l)$, $\bar{x} = (x_{l+1}, \ldots, x_n)$,
$\bar{w} = (w_1, \ldots, w_{m-n})$, $\bar{v} = (1, x_1, \ldots, x_n, w_1, \ldots, w_{m-n})$ *(same as per definition of QAP).*

– $\mathsf{SpecialG.Reprove} : (\mathsf{crs}, X', \pi', b', \mathcal{R}_\mathcal{Q}) \mapsto (X, \pi, b)$ *such that*

$$\begin{aligned} &b, r_1, r_2 \xleftarrow{\$} \mathbb{F}_p, X = X' + (b - b')K_\gamma, \pi = (O, U, V),\\ &O = \frac{1}{r_1}O', U = r_1 U' + r_1 r_2 \delta G_2, V = V' + r_2 O' - (b - b')K_\delta.\end{aligned}$$

where $\pi' = (O', U', V')$.

– $\mathsf{SpecialG.VerCom} : (\mathsf{pp}, X, \bar{x}, b) \mapsto 0/1$ *where the output is 1 iff the following holds*

$$X = \sum_{i=l+1}^{n} x_i \cdot \frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\gamma} \cdot G_1 + bK_\gamma,$$

where $\bar{x} = (x_{l+1}, \ldots, x_n)$, $0 \le l \le n - 1$.

– $\mathsf{SpecialG.Verify} : (\mathsf{crs}, \bar{y}, X, \pi, \mathcal{R}_\mathcal{Q}) \mapsto 0/1$ *where the output is 1 iff the following holds*

$$e(O, U) = e(\alpha \cdot G_1, \beta \cdot G_2) \cdot e(X + Y, \gamma \cdot G_2) \cdot e(V, \delta \cdot G_2),$$

where $\pi = (O, U, V)$, $Y = \sum_{i=1}^{l} x_i \cdot \frac{\beta a_i(\tau)+\alpha b_i(\tau)+c_i(\tau)}{\gamma} \cdot G_1$ *and* $\bar{y} = (x_1, \ldots, x_l)$.

– $\mathsf{SpecialG.Simulate} : (\mathsf{td}, \bar{y}, \mathcal{R}_{\mathcal{Q}}) \mapsto (\pi, X)$ *where* $x, o, u \overset{\$}{\leftarrow} \mathbb{F}_p$ *and let* $\pi = (o \cdot G_1, u \cdot G_2, v \cdot G_1)$ *where* $v = \frac{o \cdot u - \alpha\beta - \sum_{i=1}^{l} x_i(\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)) - x}{\delta}$ *and, by definition* $\bar{y} = (x_1, \ldots, x_l)$*. Note that* π *is a simulated proof for transparent input* $\bar{y}$ *and commitment* $X = x \cdot G_1$*.*