Ethical identity, ring VRFs, and zero-knowledge continuations

Jeffrey Burdges, Oana Ciobotaru, Handan Kılınç Alper, Alistair Stewart, and Sergey Vasilyev

Web 3.0 Foundation

May 30, 2023

Abstract. We introduce a new cryptographic primitive, named *ring verifiable random functions (ring VRF)*, which provides an array of uses, especially in anonymous credentials. Ring VRFs are ring signatures that prove correct evaluation of an authorized signer's pseudoramdom function, while hiding the signer's identity within some set of possible signers.

We present a family of ring VRF protocols with surprisingly efficient instantiations, due to our novel *zero-knowledge continuation* technique. Intuitively our ring VRF signers generate two linked proofs, an inexpensive one for PRF evaluation and one for ring membership, which can be reused. We reuse the ring membership proof across multiple inputs by expanding a Groth16 trusted setup to re-hide public inputs when rerandomizing the Groth16, ensuring that these reuses of the same proof are unlinkable. Our fastest amortized ring VRF needs only eight $\mathcal{G}_1$ and two $\mathcal{G}_2$ scalar multiplications, making it the only ring signature with performance competitive with group signatures.

A ring VRF can be used to obtain a unique pseudo-anonymous identity from a given a list of identities. By using a different input for the ring VRF in different contexts, we can generate a pseudonym for each context that is unlinkable between different contexts. We discuss applications that range across the anonymous credential space.

We define the security of ring VRFs in the universally composable (UC) model and show that our protocol is UC secure.

1 Introduction

We introduce an anonymous credential flavor called ring verifiable random functions (ring VRFs), in essence ring signatures that anonymize signers but also prove evaluation of the signers' PRFs. Ring VRFs provide a better foundation for anonymous credentials across a range of concerns, including formalization, optimizations, the nuances of use-cases, and miss-use resistance.

Along with some formalizations, we address three questions within the unfolding ring VRF story:

1. What are the cheapest SNARK proofs? Ones users reuse without reproving.
2. How can identity be safe for general use? By revealing nothing except users' uniqueness.
3. How can ration card issuance be transparent? By asking users trust a public list, not certificates.

Our contributions: We introduce new privitive, a ringVRF, and prove it's security. To efficiently implement our primitive, we present a novel techique, the Zero Knowldege continuation and define and prove its security. We discuss applications of the Ring VRF to identity, rationing and other use cases.

Ring VRFs: A ring signature [5,31,30,3,14] proves only that its actual signer lies in a "ring" of public keys, without revealing which signer really signed the message. A *verifiable random function* (VRF) is a signature that proves correct evaluation of a PRF defined by the signer's key.

A *ring verifiable random function* (ring VRF) is a ring signature, in that it anonymizes its actual signer within a ring of plausible signers, but also proves correct evaluation of a pseudo-random function (PRF) defined by the actual signer's key. Ring VRF outputs then provide linking proofs between different signatures if and only if the signatures have identical inputs, as well as pseudo-randomness.

As this pseudo-random output is uniquely determined by the signed message and signer's actual secret key, we can therefore link signatures by the same signer if and only if they sign identical messages. In effect, ring VRFs restrict anonymity similarly to but less than linkable ring signatures [30,29] do, which makes them multi-use and contextual. We define the security of ring VRFs in both the standard model and in the universally composable (UC) [11,12] model. We show that our ring VRF protocol is secure in the UC model, which allows it to be used in protocols with UC security, e.g. [7]. We build extremely efficient and flexible ring VRFs by amortizing a "zero-knowledge continuation" that unlinkably proves ring membership of a secret key, and then cheaply proving individual VRF evaluations.

Zero-knowledge continuations: Rerandomizable zkSNARKs like Groth16 [24] admit a transformation of a valid proof into another valid but unlinkable proof of the exact same statement. However, in practice, rerandomization never gets deployed because the public inputs without further processing actually link different usages, thus breaking privacy.

We demonstrate in §6 a simple transformation of any Groth16 zkSNARK into a *zero-knowledge continuation* whose public inputs involve opaque Pedersen commitments (i.e., hiding commitments), with cheaply rerandomizable blinding factors and cheaply rerandomizable proofs. These zero-knowledge continuations then prove validity of the contents of Pedersen commitments, but can now be reused arbitrarily many times, without linking the usages.

In brief, we adjust the trusted setup of the Groth16 to additionally produce an independent blinding factor base for the Groth16 public input, along with an absorbing base that cancels out this blinding factor in the Groth16 verification. As our public inputs involve opaque Pedersen commitments, they now require proofs-of-knowledge resentment of to [10].

As recursive SNARKs might remain slow, we expect zero-knowledge continuations via rerandomization become essential for zkSNARKs used in identity and elsewhere outside the crypto-currency space.

Identity uses: An identity system can be based upon ring VRFs in an natural way: After verifying an identity requesting domain name in TLS, our user agent signs into the session by returning a ring VRF signature whose input is the requesting domain name, so their ring VRF output becomes their unique identity at that domain (see §9).

At this point, our requesting domain knows each users represents distinct ring members, which prevents Sybil behavior, and permits banning specific users. At the same time, users' activities remain unlinkable across different domains

In essence, ring VRF based credentials, if correctly deployed, only prevent users being Sybil, but leak nothing more about users. We argue this yields diverse legally and ethically straightforward identity usages.

As a problematic contrast, attribute based credential schemes like IRMA ("I Reveal My Attributes") credentials [9] are being marketed as an online privacy solution, but cannot prevent users being Sybil unless they first reveal numerous attributes. Attribute based credentials therefore provide little or no privacy when used to prevent abuse.

Abuse and Sybil prevention are not merely the most common use cases for anonymous credentials, but in fact they define the "general" use cases for anonymous credentials. IRMA might improve privacy when used as "special purpose" credential in narrower situations of course, but overall attribute based credentials should *never* be considered fit for general purpose usage.

Aside from general purpose identity being problematic for attribute based credentials, our existing offline processes often better protect users' privacy and human rights than adopting online processes like IRMA. In particular, there are many proposals by the W3C for attribute based credential usage in [35], but broadly speaking they all bring matching harmful uses, as motivated below.

As an example, the W3C wants users to be able to easily prove their employment status, ostensibly so users could open bank accounts purely online. Yet, job application sites could similarly demand these same proofs of current employment, a discriminatory practice. Average users apply for jobs far more often than they open bank accounts, so credentials that prove current employment do more harm than good.

An IRMA deployment should prevent this abusive practice by making verifiers prove some legal authorization to request employment status, or other attributes, before user agents prove their attributes. Indeed IRMA deployments need to regulate IRMA verifiers, certainly by government privacy laws, or ideally by some more aggressive ethics board, but this limits their flexibility and becomes hard internationally.

Ring VRFs avoid these abuse risks by being truly unlinkable, and thus yield anonymous credentials which safely avoid legal restrictions.

Any ethical general purpose identity system should be based upon ring VRFs, not attribute based credentials like IRMA.

We credit proof-of-personhood parties by Bryan Ford, et al. [19,6] with first espousing the idea that anonymous credentials should produce contextual unique identifiers, without leaking other user attributes.

As a rule, there exist simple VRF variants for all anonymous credentials, including IRMA [9] or group signatures [32]. We focus exclusively upon ring VRFs for brevity, and because alone ring VRFs contextual linkability covers the most important use cases.

Rationing uses: A rate limiting or rationing system should provide users with a stream of single-use anonymous tokens that each enable consuming some resource. As a rule, cryptographers always construct these either from blind signatures ala [16], or else from OPRFs like PrivacyPass [18], both of which have an $O(n)$ issuance phase.

Ring VRFs yield rate limiting or rationing systems with no issuance phase: We first place into the ring the public keys for all users permitted to consume resources, perhaps all legal residents within some country. We define single-use tokens to be ring VRF signatures whose VRF input consists of a resource name, an approximate date, and a bounded counter. Now merchants reports each anonymous token back to some authority who enforces rate limits by rejecting duplicate ring VRF outputs. (See §10)

In other words, our rate limiting authority treats outputs like the “nullifiers” in anonymous payment schemes. Yet, ring VRF nullifiers need only temporarily storage, as eventually one expires the date in the VRF input. Asymptotically we thus only need $O(\texttt{users})$ storage vs the $O(\texttt{history})$ storage required by anonymous payment schemes like ZCash and blind signed tokens.

We further benefit from the “ring” credential format too, as opposed to certificate based designs like group signatures: We expect a degree of fraud whenever deploying purely certificate based systems, as witnessed by the litany of fraudulent TLS and covid certificates. Ring VRFs help mitigate fraudulent certificate concerns because the ring is a database and can be audited.

We know governments have ultimately little choice but to institute rationing in response to shortages caused by climate change, ecosystem collapse, and peak oil. Ring VRFs could help avoid ration card fraud, and thereby reduce social opposition, while also protecting essential privacy.

As an important caveat, ring VRFs need heavier verifiers than single-use tokens based on OPRFs [18] or blind signatures, but those credentials’ heavy issuance phase represents a major adoption hurdle. A ring VRF systems issue fresh tokens almost non-interactively merely by adjusting allowed VRF input on resource names, dates, and bounds. This reduces complexity, simplifies scaling, and increases flexibility.

In particular, if governments issue ration cards based upon ring VRFs then these credentials could safely support other use cases, like free tiers in online services or games, and advertiser promotions, as well as identity applications like prevention of spam and online abuse. We discuss some of these applications in §10.2.

2 Ring VRF Overview

As a beginning, we introduce the ring VRF interface, give a simple unamortized non-interactive zero-knowledge (NIZK) protocol that realizes the ring VRF properties discussed later in our UC model, and give some intuition for our later amortization trick. Similar to VRF [34], a ring VRF construction needs:

- $\mathsf{rVRF.KeyGen}$ outputs $(\mathsf{sk}, \mathsf{pk})$ algorithm, which creates a random secret key sk and associated public key pk;
- $\mathsf{rVRF.Eval} : (\mathsf{sk}, \mathsf{input}) \mapsto \mathsf{out}$ which deterministically computes the VRF output out from a secret key sk and a message input.

We demand a pseudo-randomness property from Eval. In our construction in §5, $\mathsf{rVRF.KeyGen}$ and $\mathsf{rVRF.Eval}$ resemble EC VRF like [36,37,22].

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

- $\mathsf{rVRF.Sign} : (\mathsf{sk}, \mathsf{ring}, \mathsf{input}) \mapsto \sigma$ returns a ring VRF signature σ for an input input.
- $\mathsf{rVRF.Verify} : (\mathsf{ring}, \mathsf{input}, \sigma) \mapsto \mathsf{out} \vee \perp$ returns either an output out or else failure $\perp$.

Ring VRFs differ from VRFs in that they do not expose a signer, and instead prove the signer's key lies in ring, much like how ring signatures differ from signatures. Ring VRFs differ from ring signatures in that: the verification process of Ring VRFs outputs the evaluation output out of the signer if the signature is verified with ring. So the ring signature actually proves that out is the evaluation output of the signer.

After successful verification, our verifier should be convinced that $\mathsf{pk} \in \mathsf{ring}$, that $\mathsf{out} = \mathsf{rVRF.Eval}(\mathsf{sk}, \mathsf{input})$ for some $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{rVRF.KeyGen}$. We demand anonymity meaning that the verifier learns nothing about the signer except that the signer's evaluation value of the signed message input is out and the signer's public key is in ring. In other words, this simplified ring VRF could be instantiated by making $\mathsf{rVRF.Eval}$ a pseudo-random (hash) function, and using a NIZK for a relation

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

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 input to out. Importantly, pseudo-randomness also says that out is an identity for the specific signer, but only within the context of input.

Aside from proving an evaluation using $\mathsf{rVRF.Eval}$, we need $\mathsf{rVRF.Sign}$ and $\mathsf{rVRF.Verify}$ to sign some associated data ad, as otherwise the ring VRF signature become unmoored and permits replay attacks As an example, our identity protocol below in §9 yields the same ring VRF outputs each time the same user logs into the same site, which suffers replay attacks unless ad binds the ring VRF signature to the TLS session.

Indeed, regular (non-anonymous) VRF uses always encounter similar tension with VRF inputs input being smaller than full message bodies $(\mathsf{input}, \mathsf{ad})$. As an example, Praos [17] binds their VRF public key together with a second public key for another (forward secure) signature scheme, with which they sign their ad, the block itself. An EC VRF should expose an ad parameter which it hashes when computing its challenge hashes. Aside from saving redundant signatures, exposing ad avoids user key handling mistakes that create replay attacks.

Ring VRFs cannot so easily be combined with other signatures, which makes ad essential,[1] but thankfully our ring VRF construction in §5 exposes ad exactly like EC VRFs should do.[2]

If one used the rVRF 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, ring signatures run asymptotically faster by replacing the ring argument with a set commitment to ring, roughly like what ZCash does [26]. 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 optionally 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.

We thus replace the membership condition $\mathsf{pk} \in \mathsf{ring}$ in the above relation and NIZK by the opening condition $\mathsf{pk} = \mathsf{rVRF.OpenRing}(\mathsf{comring}, \mathsf{opring})$ for some known opring.

Addressing these concerns, our notion should really be named *ring verifiable random function with additional data* and its basic methods look like

[1] If ring VRFs authorized creating blocks in an anonymous Praos blockchain then ad must include the block being created, or else others could steal their block production turn.

[2] We suppress multiple input-output pairs until §10.3 below, but they work like in [18] too.

- $\mathsf{rVRF.Sign} : (\mathsf{sk}, \mathsf{opring}, \mathsf{input}, \mathsf{ad}) \mapsto \sigma$, and
- $\mathsf{rVRF.Verify} : (\mathsf{comring}, \mathsf{input}, \mathsf{ad}, \sigma) \mapsto \mathsf{out} \vee \perp$.

Although an asymptotic improvement, our opening $\mathsf{rVRF.OpenRing}$ based condition invariably still winds up being computationally expensive to prove inside a zkSNARK. We solve this obstacle in §6 by introducing zero-knowledge continuations, a new zkSNARK technique built from rerandomizable Groth16s [24] and designed for SNARK composition and reuse.

As a step towards this, we split the relation $\mathcal{R}_{\mathsf{rvrf}}$ into a relation for rVRF evaluation and a relation, which enforces our computationally expensive condition $\mathsf{pk} = \mathsf{rVRF.OpenRing}(\mathsf{comring}, \mathsf{opring})$. We want to reuse the proof generated for latter across multiple rVRF signatures, so anonymity requires we rerandomize a Groth16 SNARK for it ala [2, Theorem 3, Appendix C, pp. 31]. Yet, we connect together the NIZKs for the two relations.

3 Preliminaries

We give definitions of some cryptographic primitives that help us to construct our ring VRF protocol.

3.1 Non-Interactive Zero-Knowledge

We let $(\mathcal{R}, z)$ denote the output of a relation generator $\mathfrak{R}$ which receives λ as an input. $\mathcal{R}$ 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:

- $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Setup}(1^\lambda) \to (crs_{\mathcal{R}}, \tau_{\mathcal{R}}, pp_{\mathcal{R}})$ given $\mathcal{R}$, it outputs a common reference string $crs_{\mathcal{R}}$, a trapdoor $\tau_{\mathcal{R}}$ and a list of public parameters $pp_{\mathcal{R}}$.
- $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Prove}(crs_{\mathcal{R}}, pp_{\mathcal{R}}, x; \omega) \mapsto \pi$ creates a proof π for $(x; \omega) \in \mathcal{R}$.
- $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Verify}(crs_{\mathcal{R}}, pp_{\mathcal{R}}, x; \pi)$ returns either true of false.
- $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Simulate}(\tau_{\mathcal{R}}, pp_{\mathcal{R}}, x) \mapsto \pi$ returns a proof π.

NIZK satisfies the following:

Definition 1. *We say* $\mathsf{NIZK}_{\mathcal{R}}$ *is* complete *if* $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Verify}(crs_{\mathcal{R}}, pp_{\mathcal{R}}, x, \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Prove}(crs_{\mathcal{R}}, pp_{\mathcal{R}}, x; \omega))$ *succeeds for all* $\lambda \in \mathbb{N}$*, for all* $(\mathcal{R}, z) \leftarrow \mathfrak{R}$*, for all* $(x; \omega) \in \mathcal{R}$*.*

Definition 2. *We say* $\mathsf{NIZK}_{\mathcal{R}}$ *is* perfect zero-knowledge *for all* $\lambda \in \mathbb{N}$*, for all* $(\mathcal{R}, z) \leftarrow \mathfrak{R}$*,* $(x, \omega) \in \mathcal{R}$ *and for all adversaries* $\mathcal{A}$*, if the following holds:*

$$\begin{aligned}
&\Pr[\mathcal{A}(crs_{\mathcal{R}}, pp_{\mathcal{R}}, z, \pi, \mathcal{R}) = 1 \mid (crs_{\mathcal{R}}, \tau_{\mathcal{R}}, pp_{\mathcal{R}}) \leftarrow \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Setup}(1^\lambda), \pi \leftarrow \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Prove}(crs_{\mathcal{R}}, pp_{\mathcal{R}}, x; \omega)] \\
=\ &\Pr[\mathcal{A}(crs_{\mathcal{R}}, pp_{\mathcal{R}}, z, \pi, \mathcal{R}) = 1 \mid (crs_{\mathcal{R}}, \tau_{\mathcal{R}}, pp_{\mathcal{R}}) \leftarrow \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Setup}(1^\lambda), \pi \leftarrow \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Simulate}(\tau_{\mathcal{R}}, pp_{\mathcal{R}}, x)]
\end{aligned}$$

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

$$\begin{aligned}
\Pr[\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Verify}(crs_{\mathcal{R}}, pp_{\mathcal{R}}, x, \pi) = 1 \wedge (x; \omega) \notin \mathcal{R} \mid (\mathcal{R}, z) \leftarrow \mathfrak{R}, \\
(crs_{\mathcal{R}}, \tau_{\mathcal{R}}, pp_{\mathcal{R}}) \leftarrow \mathsf{NIZK}_{\mathcal{R}}.\mathsf{Setup}(1^\lambda), ((x, \pi); \omega) \leftarrow (\mathcal{A}||\mathsf{Extract})(\mathcal{R}, z, crs_{\mathcal{R}}, pp_{\mathcal{R}})] = \mathsf{negl}(\lambda)
\end{aligned} \tag{1}$$

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 $crs_{\mathcal{R}}$, $pp_{\mathcal{R}}$ or $\tau_{\mathcal{R}}$ 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 $crs_{\mathcal{R}} = \emptyset$ and $pp_{\mathcal{R}} = \emptyset$. For a $\mathsf{NIZK}_{\mathcal{R}}$ such as Groth16 [24], $pp_{\mathcal{R}} = \emptyset$. However, for our particular instantiation of $\mathsf{NIZK}_{\mathcal{R}}$ in Section 6.3 with $\mathcal{R}$ defined in Section 6, the $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Setup}$ outputs all three parameters $crs_{\mathcal{R}}$, $\tau_{\mathcal{R}}$, $pp_{\mathcal{R}}$. 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. *A commitment scheme* Com *consists of two algorithms:*

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

4 Security Model of Ring VRF

In this section, we define formally a ring VRF scheme and model its security in the UC model.

Definition 5 (Ring VRF). *A ring VRF scheme is defined with public parameters pp generated by a setup algorithm* $\mathsf{rVRF.Setup}(1^\lambda)$ *and with the following PPT algorithms. We note that all algorithms below have pp as part of their input even if we do not always explicitly write it.*

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

- $\mathsf{rVRF.CommitRing}(\mathsf{ring}, \mathsf{pk}_i) \to (\mathsf{comring}, \mathsf{opring})$*: It outputs a commitment of* ring *with the opening* opring *given input* ring *and* $\mathsf{pk} \in \mathsf{ring}$.
- $\mathsf{rVRF.OpenRing}(\mathsf{comring}, \mathsf{opring}) \to \mathsf{pk}$*: It outputs a public key* pk *given commitment* $\mathsf{comring}$ *and an opening* opring *of* $\mathsf{comring}$ *to* pk.
- $\mathsf{rVRF.Sign}(\mathsf{sk}_i, \mathsf{comring}, \mathsf{opring}, \mathsf{input}, \mathsf{ad}) \to \sigma$*: It outputs a ring signature* σ *that signs a message* $\mathsf{input} \in \{0,1\}^*$ *and an associated data* $\mathsf{ad} \in \{0,1\}^*$ *with* sk_i *for* $\mathsf{comring}$.
- $\mathsf{rVRF.Verify}(\mathsf{comring}, \mathsf{input}, \mathsf{ad}, \sigma) \to (b, y)$*: It is a deterministic algorithm that outputs* $b \in \{0,1\}$ *and* $y \in \mathcal{S}_{eval} \cup \{\perp\}$*.* $b = 1$ *means verified and* $b = 0$ *means not 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.

The security properties that we want to achieve in a ring VRF scheme are informally as follows: *correctness* meaning that when an honest signer with key $(\mathsf{sk}_i, \mathsf{pk}_i)$ outputs σ by running $\mathsf{rVRF.Sign}(\mathsf{sk}_i, \mathsf{comring}, \mathsf{opring}, \mathsf{input}, \mathsf{ad})$, $\mathsf{rVRF.Verify}(\mathsf{comring}, \mathsf{input}, \mathsf{ad}, \sigma)$ must output $b = 1$ and y which is equal to $\mathsf{rVRF.Eval}(\mathsf{sk}_i, \mathsf{input})$ if and only if $\mathsf{rVRF.OpenRing}(\mathsf{comring}, \mathsf{opring}) \to \mathsf{pk}_i \in \mathsf{ring}$, *randomness* meaning that an evaluation value is random and independent from the message and the public key, *determinism* meaning that $\mathsf{rVRF.Eval}$ is deterministic, *anonymity* meaning that $\mathsf{rVRF.Sign}$ does not leak any information about its signer, *unforgeability* meaning that an adversary should not be able to forge a ring VRF signature and *uniqueness* meaning that number of verified evaluation values should not be more than the number of the keys in the ring.

We could define all these security properties formally in the standard model but this may cause composability issues when the ring VRF protocol is composed with other protocols. Considering the applications of ring VRF protocol, we want to achieve stronger security guarantees in different environments. Therefore, we define the security of a ring VRF scheme in the UC model in Section 4.1.

One can consider a ring VRF scheme is a combination of a VRF scheme and a ring signature scheme where $\mathsf{rVRF.Eval}$ is similar to Eval algorithm of a VRF scheme and $\mathsf{rVRF.Sign}$ is similar to Sign algorithm of a ring signature scheme. The subtle difference is in $\mathsf{rVRF.Verify}$ that works similar to both Verify of ring signature and VRF schemes. $\mathsf{rVRF.Verify}$ does not need the signer's public key to verify a ring VRF signature as in a ring signature scheme but it outputs the signer's evaluation value for every verified signature. If the evaluation value is generated independent from the signer's key, $\mathsf{rVRF.Verify}(\mathsf{ring}, \mathsf{input}, \mathsf{ad}, \sigma)$ does not reveal any identity of the signer except that the signer's key is in the ring.

We remark that the output of $\mathsf{rVRF.Eval}$ does not depend on a ring. Therefore, if a signer P with a public key pk signs a message input for ring where $\mathsf{pk} \in \mathsf{ring}$ and obtains σ_1 at some point and later wants to reveal its identity i.e.. σ_1 is signed by P with pk, P should resign input with another ring $\{\mathsf{pk}\}$ consisting on *only* its public key and obtain another signature σ_2. Since the verification of σ_1 and σ_2 with ring and $\{\mathsf{pk}\}$, respectively, outputs the same evaluation value, the verifier can be convinced that P signed the both signatures. Hence, a ring VRF scheme can link the identities of the signers if it is necessary.

4.1 Ring VRF in the UC Model

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 1) 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 1 in Appendix A. 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 $\mathtt{signing\_keys}$ keeps the keys of parties. The other table $\mathtt{anonymous\_key\_map}$ 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 5) 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 anonymous keys of all malicious signatures for a ring and input 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 input, $\mathcal{F}_{\mathsf{rvrf}}$ generates an anonymous key W as explained below and sets $\mathtt{anonymous\_key\_map}[\mathsf{input}, W]$ to pk. Then, it generates an evaluation value y as explained below and sets $\mathtt{evaluations}[\mathsf{input}, W]$ to y. In short, given honestly generated secret, public key pair $(\mathsf{sk}, \mathsf{pk})$ in the real world, the algorithm $\mathsf{rVRF.Eval}(\mathsf{sk}, \mathsf{input})$ that outputs evaluation value corresponds to generating an anonymous key W for $\mathsf{pk}, \mathsf{input}$ and obtaining the evaluation value stored in $\mathtt{evaluations}[\mathsf{input}, W]$ 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 (x, 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 x during signature generation of this honest party. One can imagine x as a secret key and pk as a public key. We note that it is not a problem in the ideal model that the adversary generates the secret key because a signature generated by an honest key can be valid if and only if the corresponding honest party requests it as it is guaranteed in the verification process of $\mathcal{F}_{\mathsf{rvrf}}$.

[Key Generation.] upon receiving a message $(\mathsf{keygen}, \mathsf{sid})$ from a party P_i, send $(\mathsf{keygen}, \mathsf{sid}, \mathsf{P}_i)$ to the simulator Sim. Upon receiving a message $(\mathsf{verificationkey}, \mathsf{sid}, x, \mathsf{pk})$ from Sim, verify that x, pk has not been recorded before for sid i.e., there exists no (x', pk') in $\mathtt{signing\_keys}$ such that $x' = x$ or $\mathsf{pk}' = \mathsf{pk}$. If it is the case, store in the table $\mathtt{signing\_keys}$, under P_i, the value x, pk and return $(\mathsf{verificationkey}, \mathsf{sid}, \mathsf{pk})$ to P_i.

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 $(\mathsf{corrupted}, \mathsf{sid}, \mathsf{P}_i)$.

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

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

We remark that once Sim obtains an anonymous key W of input and honest key pk, Sim can learn the evaluation of input with pk without knowing the pk via malicious ring VRF evaluation i.e., send the message $(\mathsf{eval}, \mathsf{sid}, \mathsf{pk}_i, W, \mathsf{input})$ where pk_i is a malicious verification key. Here, if W is an anonymous key of $\mathsf{input}, \mathsf{pk}$, $\mathcal{F}_{\mathsf{rvrf}}$ returns $\mathtt{evaluations}[\mathsf{input}, W]$ even if $\mathsf{pk} \neq \mathsf{pk}_i$. We note that this leakage does not contradict the desired security properties and helps us to prove our ring VRF protocol realizes $\mathcal{F}_{\mathsf{rvrf}}$.

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

[Honest Ring VRF Signature and Evaluation.] upon receiving a message $(\mathsf{sign}, \mathsf{sid}, \mathsf{ring}, \mathsf{pk}_i, \mathsf{ad}, \mathsf{input})$ 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 $\mathtt{anonymous\_key\_map}[\mathsf{input}, W'] = \mathsf{pk}_i$, let $W \leftarrow\!\!\$\ \mathcal{S}_W$ and let $y \leftarrow\!\!\$\ \mathcal{S}_{eval}$. Set $\mathtt{anonymous\_key\_map}[\mathsf{input}, W] = \mathsf{pk}_i$ and set $\mathtt{evaluations}[\mathsf{input}, W] = y$.
In any case (except ignoring), obtain W, y where $\mathtt{anonymous\_key\_map}[\mathsf{input}, W] = \mathsf{pk}_i$, $\mathtt{evaluations}[\mathsf{input}, W] = y$ and (x, pk) is in $\mathtt{signing\_keys}$. Then run $\mathsf{Gen}_{sign}(\mathsf{ring}, W, x, \mathsf{pk}, \mathsf{ad}, \mathsf{input}) \to \sigma$. Record $[\mathsf{input}, \mathsf{ad}, W, \mathsf{ring}, \sigma, 1]$. Return $(\mathsf{signature}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, \mathsf{input}, y, \sigma)$ to P_i.

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

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

Ring VRF Verification: This part of $\mathcal{F}_{\mathsf{rvrf}}$ is to check whether a ring signature of σ signs input and ad for ring with anonymous key W. This part should correspond 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 $y = \mathtt{evaluations}[\mathsf{input}, W]$. Otherwise, it outputs $b = 0$ and $y = \perp$.

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 $[\mathsf{input}, \mathsf{ad}, W, \mathsf{ring}, \sigma, b']$ 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 input (See C2). If it is the case, $\mathcal{F}_{\mathsf{rvrf}}$ checks its records whether this honest party requested signing input and ad for ring. If there exists such record i.e., $[\mathsf{input}, \mathsf{ad}, W, \mathsf{ring}, ., 1]$, 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 input and ad for ring with W once the honest party owning W has requested signing input 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 an indicator b_{Sim} showing that σ is valid or not and a public key $\mathsf{pk}_{\mathsf{Sim}}$. 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 $\mathcal{W}[\mathsf{input}, \mathsf{ring}]$ which stores the anonymous keys of valid malicious signatures of input for ring. If the number of anonymous keys in $\mathcal{W}[\mathsf{input}, \mathsf{ring}]$ 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 $\mathsf{input}, \mathsf{ring}$ 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 $\mathsf{input}, \mathsf{pk}_{\mathsf{Sim}}$ 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 $[\mathsf{input}, \mathsf{ad}, W, \mathsf{ring}, \sigma, b]$ to be able to reply with the same b for the same verification query later. If $b = 1$, $\mathcal{F}_{\mathsf{rvrf}}$ returns $\mathtt{evaluations}[\mathsf{input}, W]$ as well.

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

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

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

- if $W \notin \mathcal{W}[\mathsf{input}, \mathsf{ring}]$, add W to $\mathcal{W}[\mathsf{input}, \mathsf{ring}]$.
- if $\texttt{evaluations}[\mathsf{input}, W]$ is not defined, set $\texttt{evaluations}[\mathsf{input}, W] \leftarrow\$ \mathcal{S}_{eval}$, $\texttt{anonymous\_key\_map}[\mathsf{input}, W] = \mathsf{pk}_{\mathsf{Sim}}$. Set $\texttt{evaluations}[\mathsf{input}, W] = y$.
- otherwise, set $y = \texttt{evaluations}[\mathsf{input}, W]$.

Finally, output $(\mathsf{verified}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, \mathsf{input}, \sigma, y, b)$ 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 associated with the signer's key for the sake of anonymity. Therefore, $\mathcal{F}_{\mathsf{rvrf}}$ maps a random and independent anonymous key to each input and pk so that this key behaves as if it is the verification key of the signature. Since it is random and independent from input and pk, it does not leak any information about the signer during the verification but it still allows $\mathcal{F}_{\mathsf{rvrf}}$ to distinguish the signer.

We remark that if $\mathcal{F}_{\mathsf{rvrf}}$ records $[\mathsf{input}, \mathsf{ad}, W, \mathsf{ring}, \sigma, 1]$ in C3-3, it does not check whether $\mathsf{pk}_{\mathsf{Sim}}$ is in ring. It rather checks number of anonymous keys of malicious verified signatures for ring to make sure that the verified malicious signatures of input for ring do not output evaluation values more than malicious keys in ring to preserve the uniqueness.

$\mathcal{F}_{\mathsf{rvrf}}$ achieves the following security properties. We note that the evaluation value of $(\mathsf{input}, \mathsf{pk}_i)$ means $\texttt{evaluations}[\mathsf{input}, W]$ where $\texttt{anonymous\_key\_map}[\mathsf{input}, W] = \mathsf{pk}_i$.

Randomness: The evaluation value of $(\mathsf{input}, \mathsf{pk}_i)$ is randomly selected independent from $(\mathsf{input}, \mathsf{pk}_i)$ for keys pk_i and input's. The evaluation value of pairs $\{(\mathsf{input}, \mathsf{pk}_i)\}$ with an anonymous key W provided by Sim is randomly selected independent from $\{(\mathsf{input}, \mathsf{pk}_i)\}$ for all malicious keys pk_i and input's. We remark that since Sim can provide the same anonymous key for different public keys for a message input, we consider the randomness of an evaluation value that is generated for all pairs $\{(\mathsf{input}, \mathsf{pk}_i)\}$.

Evaluation of $(\mathsf{input}, \mathsf{pk}_i)$ where pk_i is an honest key is generated by first assigning a random anonymous key W to it and then assigning a random evaluation value y to (input, W). So, honest evaluations are always random and independent from $(\mathsf{input}, \mathsf{pk}_i)$. Malicious evaluation value of pairs $\{(\mathsf{input}, \mathsf{pk}_i)\}$ with the same anonymous W is $\texttt{evaluations}[\mathsf{input}, W]$ which is sampled randomly and independently from $\{(\mathsf{input}, \mathsf{pk}_i)\}$ by $\mathcal{F}_{\mathsf{rvrf}}$.

Determinism: The evaluation value of $(\mathsf{input}, \mathsf{pk}_i)$, once it has been evaluated, is unique and cannot be changed.

The evaluation value of $(\mathsf{input}, \mathsf{pk}_i)$ where pk_i is honest is unique and cannot be changed for honest parties because the anonymous key of it selected only once. Similarly, the evaluation value of $(\mathsf{input}, \mathsf{pk}_i)$ where pk_i is malicious is unique and cannot be changed because $\mathcal{F}_{\mathsf{rvrf}}$ does not allow Sim to select two different anonymous keys for $(\mathsf{input}, \mathsf{pk}_i)$ and to update the anonymous key.

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

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 signer having a key pk sends a message $(\mathsf{sign}, \mathsf{sid}, \mathsf{ring}, \mathsf{pk}, \mathsf{ad}, \mathsf{input})$ to $\mathcal{F}_{\mathsf{rvrf}}$. Therefore, Sim cannot create a forgery here. 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 $\mathsf{input}, \mathsf{ad}$ 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: We say that an evaluation value y for a message input is verified for ring, if there exists a signature σ such that $\mathcal{F}_{\mathsf{rvrf}}$ returns $(y, 1)$ for a query $(\mathsf{verify}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, \mathsf{input}, \sigma)$ for some anonymous key W and associated data ad. The uniqueness property guarantees that the number of verified evaluation values via signatures for a message input and ring is not more than $|\mathsf{ring}|$.

If $\mathcal{F}_{\mathsf{rvrf}}$ outputs $(1, y)$, it means that there exists a record $[\mathsf{input}, ., W, \mathsf{ring}, \sigma, 1]$ and $y = \mathtt{evaluations}[\mathsf{input}, W]$, $\mathtt{anonymous\_key\_map}[\mathsf{input}, W] = \mathsf{pk}$. 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 $\mathsf{pk} \in \mathsf{ring}$. Let's assume that there exist more verified evaluation values for ring

The evaluation value of $(\mathsf{input}, \mathsf{pk}_i)$ is unique and fixed thanks to the determinism. Uniqueness is broken if the number of verified evaluation values of input for ring is greater than the number of anonymous keys that verify a signature that signs input for ring. Assume that there exist t different verified evaluation values $\mathcal{Y} = \{y_1, y_2, \ldots, y_t\}$ of a message input for ring where $|\mathsf{ring}| = t - 1$. This implies that for each $y_i \in \mathcal{Y}$, there exists a record $[\mathsf{input}, \mathsf{ad}_i, W_i, \mathsf{ring}, \sigma_i, 1]$ such that $\mathtt{evaluations}[\mathsf{input}, W_i] = y_i$ where $\mathtt{anonymous\_key\_map}[\mathsf{input}, W_i] = \mathsf{pk}_i$ and $W_i \neq W_j$ for all $i, j \in [1, t]$. We know from the determinism property $\mathsf{pk}_i \neq \mathsf{pk}_j$ 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 $\mathcal{Y}$ maps to one honest public key in ring meaning that honest evaluation values in $\mathcal{Y}$ is at most $|\mathsf{ring} \setminus \mathsf{ring}_{mal}| = n_h$. If pk_i is not an honest verification key, $W_i \in \mathcal{W}[\mathsf{input}, \mathsf{ring}]$ since $\mathcal{F}_{\mathsf{rvrf}}$ adds W_i to $\mathcal{W}[\mathsf{input}, \mathsf{ring}]$ whenever it creates such record for a malicious signature. $\mathcal{F}_{\mathsf{rvrf}}$ makes sure that in the condition C3-2 that $\mathcal{W}[\mathsf{input}, \mathsf{ring}] \leq |\mathsf{ring}_{mal}| = n_m$. Therefore, $t \leq n_h + n_m = |\mathsf{ring}|$ which is a contradiction.

We note that Sim in our functionality can generate a valid ring signature σ that signs input with a malicious key pk for ring where the malicious key is not in ring i.e., $\mathcal{F}_{\mathsf{rvrf}}$ can have a record for a malicious signature σ such that $[\mathsf{input}, ., W_i, \mathsf{ring}, \sigma, 1]$ and $\mathtt{anonymous\_key\_map}[\mathsf{input}, W] = \mathsf{pk} \notin \mathsf{ring}$. However, it cannot create signatures of input for ring which $\mathcal{F}_{\mathsf{rvrf}}$ verifies and outputs more than $|\mathsf{ring}_{mal}|$ different evaluation values.

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: An honest signature σ that signs $\mathsf{input}, \mathsf{ad}$ and verified by a ring and anonymous key W does not give any information about its signer except that its key is in ring if input is not signed by the same signer before for any other ring. We define this formally with the anonymity game below. We note that we cannot define and verify this property in $\mathcal{F}_{\mathsf{rvrf}}$ like we did for the other properties because it depends on how Gen_{sign} is defined.

Definition 6 (Anonymity). *We define the anonymity game against a special environment $\mathcal{D}$ which plays the following anonymity game. $\mathcal{F}_{\mathsf{rvrf}}$ satisfies anonymity, if any PPT distinguisher $\mathcal{D}$ has a negligible advantage in λ to win the anonymity game defined as follows: We define the anonymity game between a challenger and $\mathcal{D}$. $\mathcal{D}$ accesses a signing oracle $\mathcal{O}_{Sign}$ and $\mathcal{F}_{\mathsf{rvrf}}$ simulated by the challenger as described in Figure 1.*

- *Given the input $'\mathsf{keygen}'$, $\mathcal{O}_{Sign}$ sends $(\mathsf{keygen}, \mathit{sid})$ to the challenger and obtains a verification key pk. Then, it stores pk to a list $\mathcal{K}$ and outputs pk.*
- *Given the input $'(\mathsf{pk}, \mathsf{ring}, \mathsf{ad}, \mathsf{input})'$, $\mathcal{O}_{Sign}$ sends $(\mathsf{sign}, \mathit{sid}, \mathsf{ring}, \mathsf{pk}, \mathsf{ad}, \mathsf{input})$ to the challenger and receives $(\mathsf{signature}, \mathit{sid}, \mathsf{ring}, W, \mathsf{ad}, \mathsf{input}, y, \sigma)$ if $\mathsf{pk} \in \mathsf{ring}$. Then $\mathcal{O}_{Sign}$ stores input to a list $\mathsf{signed}[\mathsf{pk}]$. It outputs (σ, W). Otherwise, it outputs $\perp$.*

At some point, $\mathcal{D}$ sends $(\mathsf{ring}, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{input}, \mathsf{ad})$ to the challenger where $\mathsf{pk}_0, \mathsf{pk}_1 \in \mathsf{ring}$, $\mathsf{input} \notin \mathsf{signed}[\mathsf{pk}_0]$ and $\mathsf{input} \notin \mathsf{signed}[\mathsf{pk}_1]$[3]. Challenger lets $b \leftarrow_r \{0,1\}$. Then it gives the input $(\mathsf{pk}_b, \mathsf{ring}, \mathsf{input}, \mathsf{ad})$ to $\mathcal{O}\mathsf{Sign}$ and receives either $\perp$ or (σ, W). If it is (σ, W), it sends (σ, W) to $\mathcal{D}$ as a challenge. If $\mathcal{D}$ sends $'(\mathsf{pk}, \mathsf{ring}, \mathsf{ad}, \mathsf{input})'$ to $\mathcal{O}_{Sign}$ where $\mathsf{pk} = \mathsf{pk}_0$ or $\mathsf{pk} = \mathsf{pk}_1$, it loses the game. During the game if $\mathcal{D}$ outputs $b' = b$, $\mathcal{D}$ wins.

5 Ring VRF construction

In the following we instantiate our ring VRF construction with an efficient evaluation proof, which we call the Pedersen VRF and denote PedVRF. PedVRF instantiates the NIZK for the relation $\mathcal{R}_{\mathtt{eval}}$ introduced in a general form in §2. In this section we focus upon Pedersen VRF and the relations describing a SNARK for ring membership; we discuss the zero-knowledge continuation that makes the overall ring VRF efficient in the next section.

Our construction works with public parameters $pp_{rvrf} = (crs_{rvrf}, p, \mathbf{G}, G, K, \mathcal{S}_{eval} = \mathbb{F}_p)$ generated by $\mathsf{rVRF.Setup}(1^\lambda)$. Here, p is a prime number and the order of the group $\mathbf{G}$ with generators G, K. crs_{rvrf} is a common reference string generated by $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{ring}}}.\mathsf{Setup}(1^\lambda)$. Our ring VRF construction deploys random oracles $H_p, H : \{0,1\}^* \to \mathbb{F}_p$, $H_{\mathbf{G}} : \{0,1\}^* \to \mathbf{G}$ and H_{ring} for constructing a Merkle tree.

We describe Pedersen VRF before our ring VRF construction.

Pedersen VRF: We construct PedVRF similarly to [36,37,22], except we replace the public key by a Pedersen commitment $\mathsf{sk}\, G + \mathsf{b}\, K$ to the secret key sk. We do not expose a public key from KeyGen, nor inject the public key in Eval.

- $\mathsf{PedVRF.KeyGen} : pp_{rvrf} \mapsto \mathsf{sk}$ where $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{F}_p$.
- $\mathsf{PedVRF.Eval} : (\mathsf{sk}, \mathsf{input}) \mapsto H(\mathsf{input}, \mathsf{preout})$ where $\mathsf{preout} = \mathsf{sk}\, H_{\mathbf{G}}(\mathsf{input})$.
- $\mathsf{PedVRF.CommitKey} : \mathsf{sk} \mapsto (b, \mathsf{compk})$ where $\mathsf{b} \leftarrow\!\!\$\ \mathbb{F}_p$ is a blinding factor and $\mathsf{compk} = \mathsf{sk}\, G + \mathsf{b}\, K$ is a Pedersen commitment.

Our Sign and Verify algorithms of PedVRF correspond to the Prove algorithm and verification procedure of a Chaum-Pedersen DLEQ proof for relation $\mathcal{R}_{eval}$ (see below), instantiated by a Fiat-Shamir transform of a sigma protocol.

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

- $\mathsf{PedVRF.Sign} : (\mathsf{sk}, \mathsf{b}, \mathsf{input}, \mathsf{ad}) \mapsto \sigma$. Compute $\mathsf{preout} := \mathsf{sk}\, H_{\mathbf{G}}(\mathsf{input})$ and $\mathsf{compk} = \mathsf{sk}G + \mathsf{b}K$. Then, run $\mathsf{NIZK}_{\mathcal{R}_{eval}}.\mathsf{Prove}(\mathsf{compk}, \mathsf{preout}, \mathsf{input}; \mathsf{sk}, \mathsf{b}, \mathsf{input})$ 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 $R = r_1 G + r_2 K, R_m = r_1 H_{\mathbf{G}}(\mathsf{input})$ and $c = H_p(\mathsf{ad}, \mathsf{input}, \mathsf{compk}, \mathsf{preout}, R, R_m)$, finally compute $s_1 = r_1 + c\,\mathsf{sk}$ and $s_2 = r_2 + c\,\mathsf{b}$ and let $\pi = (c, s_1, s_2)$. Return the signature $\sigma = (\mathsf{preout}, \pi)$.

[3] The challenger needs to check this because if input is signed before by one of $\mathsf{pk}_0, \mathsf{pk}_1$, then the anonymity is broken trivially because verification of any signature of input for different rings outputs the same evaluation value

– $\mathsf{PedVRF.Verify} : (\mathsf{compk}, \mathsf{input}, \mathsf{ad}, \sigma) \mapsto \mathsf{out} \vee \perp$. It runs $\mathsf{NIZK}_{\mathcal{R}_{eval}}.\mathsf{Verify}((\mathsf{compk}, \mathsf{preout}, \mathsf{input}), \pi_{eval})$ i.e., Parse $\sigma = (\mathsf{preout}, c, s_1, s_2)$ and check if $c = H_p(\mathsf{ad}, \mathsf{input}, \mathsf{compk}, \mathsf{preout}, R, R_m)$ where $R = s_1 G + s_2 K - c\,\mathsf{compk}$ and and $R_m = s_1 H_{\mathbf{G}}(\mathsf{input}) - c\,\mathsf{preout}$, then return $H(\mathsf{input}, \mathsf{preout})$, which equals $\mathsf{PedVRF.Eval}(\mathsf{sk}, \mathsf{input})$, or return failure $\perp$ otherwise.

We remark that PedVRF becomes almost EC VRF if we demand $\mathsf{b} = r_2 = 0$ in Sign.

The Ring VRF Construction: We now describe our ring VRF construction as a combination of PedVRF, $\mathsf{NIZK}_{\mathcal{R}_{\mathtt{ring}}}$ (for a relation $\mathcal{R}_{\mathtt{ring}}$ instantiated below) and a commitment scheme Com.

– $\mathsf{rVRF.KeyGen}$ returns as secret key $\mathsf{sk}, r \leftarrow\!\!\$\, \mathbb{F}_p$ and pk as public key where $\mathsf{pk} = \mathsf{Com.Commit}(\mathsf{sk}, r)$. We note that pk can be alternatively defined as $\mathsf{pk} = \mathsf{sk}G$ according to the SNARK used for $\mathcal{R}_{\mathtt{ring}}$. In this case, we would not have r as a part of the secret key. We provide one optimal public key design in §6.3 for our SNARK used for $\mathcal{R}_{\mathtt{ring}}$.
– $\mathsf{rVRF.Eval}((\mathsf{sk}, r), \mathsf{input})$ runs $\mathsf{PedVRF.Eval}(\mathsf{sk}, \mathsf{input})$. We remark that the evaluation value is generated with *only* the first part of the secret key which is sk.
– $\mathsf{rVRF.CommitRing} : (\mathsf{ring}, \mathsf{pk}) \mapsto (\mathsf{comring}, \mathsf{opring})$ computes a Merkle tree root $\mathsf{comring}$ using H_{ring} considering the elements of ring as leaves and generating a Merkle tree path opring that verifies $\mathsf{pk} \in \mathsf{ring}$.
– $\mathsf{rVRF.OpenRing} : (\mathsf{comring}, \mathsf{opring}) \mapsto \mathsf{pk}$ verifies that the root computed via Merkle tree path opring is $\mathsf{comring}$. If it is the case, output $\mathsf{pk} \in \mathsf{opring}$. Otherwise, output $\perp$.
We choose the ring commitment scheme so the $\mathsf{rVRF.OpenRing}$ invocation is relatively SNARK friendly in our ring membership relation $\mathcal{R}_{\mathtt{ring}}$. We note that an alternative ring commitment scheme may be used where $\mathsf{comring} = \mathsf{ring}$ and $\mathsf{opring} = \mathsf{pk}$.

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:

– $\mathsf{rVRF.Sign} : ((\mathsf{sk}, r), \mathsf{comring}, \mathsf{opring}, \mathsf{input}, \mathsf{ad}) \mapsto \rho$ returns a ring VRF signature $\rho = (\mathsf{compk}, \pi_{\mathtt{ring}}, \sigma, \mathsf{comring})$. In this, $(\mathsf{b}, \mathsf{compk}) \leftarrow \mathsf{PedVRF.CommitKey}(\mathsf{sk})$, $\pi_{\mathtt{ring}} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{\mathtt{ring}}}.\mathsf{Prove}((\mathsf{compk}, \mathsf{comring}); \mathsf{b}, \mathsf{opring}, \mathsf{pk}, \mathsf{sk}, r)$ where $\mathsf{ad}' \leftarrow \mathsf{ad} \mathbin{+\!\!+} \pi_{\mathtt{ring}} \mathbin{+\!\!+} \mathsf{comring}$, $\sigma \leftarrow \mathsf{PedVRF.Sign}(\mathsf{sk}, \mathsf{b}, \mathsf{input}, \mathsf{ad}')$ and
$$\mathcal{R}_{\mathtt{ring}} = \left\{ (\mathsf{compk}, \mathsf{comring}; \mathsf{b}, \mathsf{opring}, \mathsf{sk}, r) \;\middle|\; \begin{array}{l} \mathsf{pk} = \mathsf{OpenRing}(\mathsf{comring}, \mathsf{opring}), \\ \mathsf{sk} = \mathsf{Com.Open}(\mathsf{pk}, r), \\ \mathsf{compk} = \mathsf{sk}G + \mathsf{b}K \end{array} \right\}$$
We note that if $\mathsf{pk} = \mathsf{sk}G$ then $\mathcal{R}_{\mathtt{ring}}$ does not need sk, r as a part of the witness. In this case we need to replace the last two conditions by $\mathsf{compk} = \mathsf{pk} + \mathsf{b}K$.
– $\mathsf{rVRF.Verify} : (\mathsf{comring}, \mathsf{input}, \mathsf{ad}, \rho) \mapsto (1, \mathsf{out}) \vee (0, \perp)$ parses ρ as $(\mathsf{compk}, \pi_{\mathtt{ring}}, \sigma)$, 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 $\mathsf{PedVRF.Verify}(\mathsf{compk}, \mathsf{input}, \mathsf{ad}', \sigma)$.

Appendix A proves our ring VRF construction realizes $\mathcal{F}_{\mathsf{rvrf}}$ in Figure 1 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 $H_{\mathbf{G}}$. 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 $\mathbf{G}$ (Lemma 5) so that preout is indistinguishable from a random element in $\mathbf{G}$. The unforgeability and uniqueness come from the fact that CDH is hard in $\mathbf{G}$ (Lemma 6).

Theorem 1. rVRF *over* pp_{rvrf} *realizes* $\mathcal{F}_{\mathsf{rvrf}}$ *running* Gen_{sign} *in Algorithm 1 [11,12] in the random oracle model assuming that* $\mathsf{NIZK}_{\mathcal{R}_{eval}}$ *and* $\mathsf{NIZK}_{\mathcal{R}_{ring}}$ *are zero-knowledge and knowledge sound, the decisional Diffie-Hellman (DDH) problem are hard in* $\mathbf{G}$ *(so the CDH problem is hard as well) and the commitment scheme* Com *is binding and perfectly hiding.*

Algorithm 1 $\mathsf{Gen}_{sign}(\mathsf{ring}, W, x, \mathsf{pk}, \mathsf{ad}, \mathsf{input})$

```
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: $\pi_{ring} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{\mathsf{ring}}}.\mathsf{Prove}(crs, \mathsf{comring}, \mathsf{compk}\ \mathsf{b}, \mathsf{opring}, x)$
7: return $\sigma = (\pi_{eval}, \pi_{ring}, \mathsf{compk}, \mathsf{comring}, W)$
```

We have the security proof of Theorem 6 in Appendix A. We give a proof sketch here.
Proof Sketch: We first verify that $\mathcal{F}_{\mathsf{rvrf}}$ running Gen_{sign} gives the anonymity defined in Definition 6 (See Lemma 4). We can show it by reducing the anonymity game in another game where we replace $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{ring}}}.\mathsf{Prove}(\mathsf{comring}, \mathsf{compk}\ \mathsf{b}, \mathsf{opring}, x)$ by $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{ring}}}.\mathsf{Simulate}(\mathsf{comring}, \mathsf{compk})$ in Gen_{sign}. Since the signature generated in this reduction is independent from honest keys thanks to perfectly hiding compk, the signatures generated by Gen_{sign} are indistinguishable.

After making sure that $\mathcal{F}_{\mathsf{rvrf}}$ running Gen_{sign} satisfies all security properties, we show that our protocol rVRF realizes $\mathcal{F}_{\mathsf{rvrf}}$. So, we construct a simulator Sim that simulates the honest parties in the real protocol rVRF and simulates the adversary in the ideal world with $\mathcal{F}_{\mathsf{rvrf}}$. Sim simulates the random oracles against adversaries in the real protocol. It simulates $H_{\mathbf{G}}$ so that it knows the discrete logarithm of each $H_{\mathbf{G}}$ output. This trapdoor helps Sim to learn the public key of the adversary if the adversary ever comes to the random oracle H to compute the evaluation value of input. When it happens, Sim asks $\mathcal{F}_{\mathsf{rvrf}}$ to evaluate input for this public key and replies the evaluation value provided by $\mathcal{F}_{\mathsf{rvrf}}$ as a random oracle H. Thus, Sim makes the evaluation values of real world malicious parties be consistent with the evaluation values of ideal world malicious parties. Whenever $\mathcal{F}_{\mathsf{rvrf}}$ sends a message $(\mathsf{verify}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, \mathsf{input}, \sigma)$ to Sim, Sim runs $\mathsf{rVRF.Verify}(\mathsf{comring}, \mathsf{input}, \mathsf{ad}, \sigma) \to (b_{\mathsf{Sim}}, .)$ and sends a response including b_{Sim}. If $\mathcal{F}_{\mathsf{rvrf}}$ replies with a decision $b \neq b_{\mathsf{Sim}}$, Sim aborts because of the inconsistency. Then, we make sure that our simulation is indistinguishable. We show this in Lemma 5 and 6 in Appendix A: We show that the indistinguishability of outputs in Lemma 5. Differently than ideal honest parties, the honest parties in our protocol have preout as a part of their signature. The ideal world honest parties has W instead of it which is sampled randomly by $\mathcal{F}_{\mathsf{rvrf}}$ instead of it. We can verify by the DDH assumption that W and preout are indistinguishable. We also show that Sim does not abort during the simulation except with negligible probability in Lemma 6 in Appendix A under the assumption of the hardness of the CDH problem.

6 Zero-knowledge Continuations

In the following, we describe a NIZK for a relation $\mathcal{R}$ where

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

and $\mathcal{R}_1$, $\mathcal{R}_2$ are NP relations. At a high level, this is based on the commit-and-prove methodology [28,13,10] as relations $\mathcal{R}_1$ and $\mathcal{R}_2$ have input $\bar{x}$ in common and the respective proofs or arguments can and will make use of a commitment X to $\bar{x}$. However, our proposed NIZK has a more specific functionality: it is designed to efficiently re-prove membership for relation $\mathcal{R}_1$ via a new technique which we call *zero-knowledge continuation.* In practice, using a NIZK that ensures a zero-knowledge continuation for a subcomponent relation (i.e., in our case $\mathcal{R}_1$) means one essentially needs to create only once an otherwise expensive proof for that subcomponent relation; the initial proof can later be re-used multiple times (just after inexpensive re-randomisations) while preserving knowledge soundness and zero-knowledge of the entire NIZK. Below, we formally define zero-knowledge continuation. In section 6.1 we instantiate it via a *special(ised) Groth16* or SpecialG, and finally, in section 6.3 we use it to instantiate our rVRF.Sign algorithm in Section 5 with fast amortised prover time.

In addition, the anonymity property of our ring VRF demands we not only finalise multiple times a component of the zero-knowledge continuation but also each time the result remains unlinkable to previous finalisations, meaning our ring VRF stays zero-knowledge even with a continuation component being reused. We formalise such a more general zero-knowledge property in section 6.1 and give an instantiation of our NIZK fulfilling such a property in section 6.3.

Definition 7 (ZK Continuations). *A zero-knowledge continuation* ZKCont *for a relation $\mathcal{R}_1$ with input $(\bar{y}, \bar{x})$ and witness $\bar{w}_1$ is a tuple of efficient algorithms (*ZKCont.Setup, ZKCont.Preprove, ZKCont.Reprove, ZKCont.VerCom, ZKCont.Verify, ZKCont.Simulate*) such that for implicit security parameter λ,*

- ZKCont.Setup : $(1^\lambda, \mathcal{R}_1) \mapsto (crs, td, pp)$ *a setup algorithm that on input the security parameter outputs a common reference string crs, a trapdoor td and a list pp of public parameters,*
- ZKCont.Preprove : $(crs, \bar{y}, \bar{x}, \bar{w}_1, \mathcal{R}_1) \mapsto (X', \pi', b')$ *constructs commitment X' from a vector of inputs $\bar{x}$ (called* opaque*) and constructs proof π' from vector of inputs $\bar{y}$ (called* transparent*), $\bar{x}$ and vector of witnesses $\bar{w}_1$, and it also outputs b' as the opening for X',*
- ZKCont.Reprove : $(crs, X', \pi', b', \mathcal{R}_1) \mapsto (X, \pi, b)$ *returns a new commitment X and proof π and returns an opening b for the commitment,*
- ZKCont.VerCom : $(pp, X, \bar{x}, b) \mapsto 0/1$ *verifies that indeed X is a commitment to $\bar{x}$ with opening (e.g., randomness) b and outputs 1 if indeed that is the case and 0 otherwise,*
- ZKCont.Verify : $(crs, \bar{y}, X, \pi, \mathcal{R}_1) \mapsto 0/1$ *outputs 1 in case it accepts and 0 otherwise,*
- ZKCont.Simulate : $(td, \bar{y}, \mathcal{R}_1) \mapsto (\pi, X)$ *takes as input a simulation trapdoor td and statement $(\bar{y}, \bar{x})$ and returns arguments π and X,*

and satisfies perfect completeness for Preprove *and, for* Reprove*, it satisfies knowledge soundness and zero-knowledge as defined below:*

We define perfect completeness for Preprove *and* Reprove *algorithms separately and in the most general way possible, (i.e., with inputs supplied by the adversary where possible).*
Perfect Completeness for Preprove *For all $\lambda \in \mathbb{N}$, for every $(\bar{y}, \bar{x}; \bar{w}_1) \in \mathcal{R}_1$:*

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

Perfect Completeness for Reprove *For all $\lambda \in \mathbb{N}$, for every PPT adversary A:*

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

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

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

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. It allows us to formalise the intuition that if one calls once ZKCont.Preprove *on $((\bar{y}, \bar{x}), \bar{w}_1) \in \mathcal{R}_1$ and then sequentially calls* ZKCont.Reprove *at least once but also (possibly) multiple times, then every time after the first call to* ZKCont.Reprove *the resulting proofs reveal nothing regarding either $\bar{x}$ or $\bar{w}_1$. Hence, the proofs obtained via sequential use of* ZKCont.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* $\lambda \in \mathbb{N}$, *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* A *there exists a PPT algorithm* Simulate *such that:*

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

6.1 Specialised Groth16

Below we instantiate our zero-knowledge continuation notion with a scheme based on Groth16 [24] SNARK; hence, we call our instantiation *specialised Groth16* or SpecialG. First, we remind the reader the definition of a Quadratic Arithmetic Program (QAP) [10], [21] and related $\mathcal{R}_\mathcal{Q}$.

Definition 8 (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* $\bar{w} = (w_1, \ldots, w_{m-n})$.

In summary, SpecialG setup is an extension of original Groth16 [24] setup by two additional group elements. SpecialG setup is identical to commit-carrying LegoSNARK ccGro16 [10, Fig. 22] setup[4]. Moreover, together the Preprove and the Reprove procedures of SpecialG are identical to the proving procedure in ccGro16 with the difference that SpecialG also re-randomises the commitment to part of the public input, as part of Reprove. Additionally, the verification procedure and the commitment verification are identical to their counterparts in ccGro16.

Let $\mathbb{G}$ be a pairing friendly elliptic curve, let e be its associated secure, efficient and non-degenerate pairing and given its related first and second source groups (with randomly chosen generators g_1 and g_2, respectively) and its target group, we introduce

Definition 9 (Specialised Groth16 (SpecialG)). *Let* $\mathcal{R}_\mathcal{Q}$ *be as mentioned above. We call specialised Groth16 for relation* $\mathcal{R}_\mathcal{Q}$ *the following:*

- $\mathsf{SpecialG.Setup} : (1^\lambda, \mathcal{R}_\mathcal{Q}) \mapsto (crs, td, pp)$.
  Let $\alpha, \beta, \gamma, \delta, \tau, \eta \xleftarrow{\$} \mathbb{F}_q^*$. *Let* $td = (\alpha, \beta, \gamma, \delta, \tau, \eta)$.
  Let $crs = (\bar{\sigma}_1, \bar{\sigma}_2)$ *where*

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

[4] ccGro16 is, in turn, based on Groth16.

$pp = \left(\left\{\frac{\beta a_i(\tau)+\alpha b_i(\tau)+c_i(\tau)}{\gamma}\cdot \mathsf{g}_1\right\}_{i=l+1}^{n}, \frac{\eta}{\gamma}\cdot \mathsf{g}_1\right)$.
Moreover, for simplicity and later use, we call $K_\gamma = \frac{\eta}{\gamma}\cdot \mathsf{g}_1$ *and* $K_\delta = \frac{\eta}{\delta}\cdot \mathsf{g}_1$.

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

$$\begin{aligned}
&b' = 0; r, s \xleftarrow{\$} \mathbb{F}_p; X' = \sum_{i=l+1}^{n} v_i \cdot \frac{\beta a_i(\tau)+\alpha b_i(\tau)+c_i(\tau)}{\gamma}\cdot \mathsf{g}_1;\\
&o = \alpha + \sum_{i=0}^{m} v_i\cdot a_i(\tau) + r\cdot\delta; u = \beta + \sum_{i=0}^{m} v_i \cdot b_i(\tau) + s\cdot \delta;\\
&v = \frac{\sum_{i=n+1}^{m}(v_i(\beta a_i(\tau)+\alpha b_i(\tau)+c_i(\tau))) + h(\tau)t(\tau)}{\delta} + o\cdot s + u\cdot r - r\cdot s\cdot\delta;\\
&\pi' = (o\cdot \mathsf{g}_1, u\cdot \mathsf{g}_2, v\cdot \mathsf{g}_1),
\end{aligned}$$

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

$$\begin{aligned}
&b, r_1, r_2 \xleftarrow{\$} \mathbb{F}_p, X = X' + (b-b')K_\gamma, \pi = (O, U, V),\\
&O = \frac{1}{r_1}O', U = r_1U' + r_1r_2\delta \mathsf{g}_2, V = V' + r_2O' - (b-b')K_\delta.
\end{aligned}$$

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

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

$$X = \sum_{i=l+1}^{n} x_i \cdot \frac{\beta a_i(\tau)+\alpha b_i(\tau)+c_i(\tau)}{\gamma}\cdot \mathsf{g}_1 + bK_\gamma,$$

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

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

$$e(O, U) = e(\alpha\cdot \mathsf{g}_1, \beta\cdot \mathsf{g}_2)\cdot e(X+Y, \gamma\cdot \mathsf{g}_2)\cdot e(V, \delta\cdot \mathsf{g}_2),$$

where $\pi = (O, U, V)$, $Y = \sum_{i=1}^{l} x_i \cdot \frac{\beta a_i(\tau)+\alpha b_i(\tau)+c_i(\tau)}{\gamma}\cdot \mathsf{g}_1$ *and* $\bar{y} = (x_1, \ldots, x_l)$.

– SpecialG.Simulate : $(td, \bar{y}, \mathcal{R}_{\mathcal{Q}}) \mapsto (\pi, X)$ *where* $x, o, u \xleftarrow{\$} \mathbb{F}_p$ *and let* $\pi = (o\cdot \mathsf{g}_1, u\cdot \mathsf{g}_2, v\cdot \mathsf{g}_1)$ *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* $X = x\cdot \mathsf{g}_1$.

Finally, we are ready to prove that SpecialG is a zero-knowledge continuation. We show that the knowledge soundness property of SpecialG (i.e., as defined for ZKCont) 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 ZKCont 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 [20],* SpecialG *is a zero-knowledge continuation as per Definition 7.*

Proof. It is straightforward to show that SpecialG has perfect completeness for Preprove and perfect completeness for Reprove.

We prove knowledge-soundness (KS) an in Definition 7 by first arguing SpecialG is a cc-SNARK with double binding (see Definition 3.4 [10]). We use the fact that 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 A_{SpecialG} be an adversary for KS in Definition 7 and define adversary A_{ccGro16} for KS in Definition 3.4 [10]:

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

Given extractor E_{ccGro16} fulfilling Definition 3.4 [10] for A_{ccGro16}, we construct extractor E_{SpecialG} for A_{SpecialG}

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

We show E_{SpecialG} fulfils Definition 7 for A_{SpecialG}. Assume by contradiction that is not the case. This implies there exists an auxiliary input aux such that each:

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

hold 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 E_{ccGro16} is an extractor for A_{ccGro16} as per Definition 3.4 [10], then each of the two events

$$\mathsf{ccGro16}.\textit{VerCommit}^*(ck, X, \bar{x}^*, b^*) = 1 \; (40) \; ; \; (\bar{y}, \bar{x}^*; \bar{w}^*) \in \mathcal{R}_{\mathcal{Q}} \; (50)$$

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 7 is false.

Finally, regarding zero-knowledge, it is clear that if $\pi = (O, U, V)$ is part of the output of $\mathsf{SpecialG.Reprove}$, then O and U 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 $\mathsf{SpecialG}$.

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

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

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

with $\mathcal{R}_2$ from preamble of Section 6. Then we define the system $\mathsf{NIZK}_{\mathcal{R}}$ for relation $\mathcal{R}$ from the preamble of this section as:

- $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Setup} : (1^\lambda) \mapsto (crs_{\mathcal{R}} = (crs, crs_{\mathcal{R}_2'(pp)}), td_{\mathcal{R}} = (td, td_{\mathcal{R}_2'(pp)}), pp_{\mathcal{R}} = pp)$ where $(crs, td, pp) \leftarrow \mathsf{ZKCont.Setup}(1^\lambda, \mathcal{R}_1)$, $(crs_{\mathcal{R}_2'(pp)}, td_{\mathcal{R}_2'(pp)}) \leftarrow \mathsf{NIZK}_{\mathcal{R}_2'(pp)}.\mathsf{Setup}(1^\lambda)$.
- $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Prove} : (crs_{\mathcal{R}}, \bar{y}, \bar{z}; \bar{x}, \bar{w}_1, \bar{w}_2) \mapsto (\pi_1, \pi_2, X)$ where $(X', \pi_1', b') \leftarrow \mathsf{ZKCont.Preprove}(crs, \bar{y}, \bar{x}, \bar{w}_1, \mathcal{R}_1)$, $(X, \pi_1, b) \leftarrow \mathsf{ZKCont.Reprove}(crs, X', \pi_1', b', \mathcal{R}_1)$, $\pi_2 \leftarrow \mathsf{NIZK}_{\mathcal{R}_2'(pp)}.\mathsf{Prove}(crs_{\mathcal{R}_2'(pp)}, X, \bar{z}; \bar{x}, b, \bar{w}_2)$.

– $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Verify} : (crs_{\mathcal{R}}, (\bar{y}, \bar{z}), (\pi_1, \pi_2, X)) \mapsto 0/1$ where the output is 1 iff

$$\mathsf{ZKCont.Verify}(crs, \bar{y}, X, \pi_1, \mathcal{R}_1) = 1 \ \wedge \ \mathsf{NIZK}_{\mathcal{R}_2'(pp)}.\mathsf{Verify}(crs_{\mathcal{R}_2'(pp)}, X, \bar{z}, \pi_2) = 1.$$

– $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Simulate} : (td_{\mathcal{R}}, \bar{y}, \bar{z}) \mapsto (\pi_1, \pi_2, X)$ where
$(\pi_1, X) \leftarrow \mathsf{ZKCont.Simulate}(td, \bar{y}, \mathcal{R}_1)$, $\pi_2 \leftarrow \mathsf{NIZK}_{\mathcal{R}_2'(pp)}.\mathsf{Simulate}(td_{\mathcal{R}_2'(pp)}, X, \bar{z})$.

Lemma 1 (Knowledge-soundness for $\mathsf{NIZK}_{\mathcal{R}}$). *If* ZKCont *is a zk continuation for $\mathcal{R}_1$ and $\mathsf{NIZK}_{\mathcal{R}_2'(pp)}$ is a NIZK for $\mathcal{R}_2'(pp)$ 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 ZKCont and $\mathsf{NIZK}_{\mathcal{R}_2'(pp)}$ due to their respective knowledge-soundness.

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

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

Lemma 2 (Special Perfect Completeness). *If* ZKCont *is a zk continuation for $\mathcal{R}_1$ and $\mathsf{NIZK}_{\mathcal{R}_2'(pp)}$ is a NIZK for $\mathcal{R}_2'(pp)$ 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 $\mathsf{NIZK}_{\mathcal{R}_2'(pp)}$ and perfect completeness for ZKCont.Reprove.

Finally, we define

Zero-knowledge after Reproving a ZKCont 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:

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

Lemma 3 (ZK after Reproving a ZKCont Proof). *If* ZKCont *is a zk continuation for $\mathcal{R}_1$ and $\mathsf{NIZK}_{\mathcal{R}_2'(pp)}$ is a NIZK for $\mathcal{R}_2'(pp)$ for some appropriately chosen public parameters pp, then the $\mathsf{NIZK}_{\mathcal{R}}$ construction described above has zero-knowledge after reproving a* ZKCont *proof.*

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

Corollary 1. *If* ZKCont *is a zk continuation for $\mathcal{R}_1$ and $\mathsf{NIZK}_{\mathcal{R}_2'(pp)}$ is a NIZK for $\mathcal{R}_2'(pp)$ 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 1, Lemma 2, Lemma 3 and we obtain the above statement.

6.3 Our Ring VRF Construction based on ZKCont

We modify slightly the signing and verification algorithms of our construction in Section 5 by deploying ZKCont. The new protocol enjoys the rerandomization properties of ZKCont which lets a signer sign a different input for the same ring without running the heavier part of prove algorithm of a NIZK related to showing the key is in the ring.

- rVRF.Setup(1^λ) outputs $pp_{rvrf} = (crs, pp = (p, \mathbf{G}, G, K), \mathbb{F}_p)$ where pp is the output of ZKCont.Setup($1^\lambda, \mathcal{R}^{\texttt{inner}}_{\texttt{ring}}$) and $\mathcal{R}^{\texttt{inner}}_{\texttt{ring}}$ is defined below.
- rVRF.Sign : $((\mathsf{sk}, r), \mathsf{comring}, \mathsf{opring}, \mathsf{input}, \mathsf{ad}) \mapsto \sigma$ generates $\mathsf{preout} = \mathsf{sk} H_{\mathbf{G}}(\mathsf{input})$, lets $\mathsf{out} = H(\mathsf{input}, \mathsf{preout})$. Then runs $\mathsf{NIZK}_{\mathcal{R}_{\texttt{rvrf}}}.\mathsf{Prove}(\mathsf{comring}, \mathsf{preout}, \mathsf{input}, \mathsf{out}; \mathsf{sk}, r, \mathsf{opring}) \to \pi_{rvrf}$ where

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

and

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

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

  We instantiate $\mathsf{NIZK}_{\mathcal{R}_{\texttt{rvrf}}}.\mathsf{Prove}$ as described in Section 6.2 where $\mathcal{R}_1 = \mathcal{R}^{\texttt{inner}}_{\texttt{ring}}$ and $\mathcal{R}'_2(pp) = \mathcal{R}_{eval}$ and $\mathcal{R} = \mathcal{R}_{\texttt{rvrf}}$. It works as follows: It runs ZKCont.Preprove(crs, comring, sk, $(r, \mathsf{opring}), \mathcal{R}^{\texttt{inner}}_{\texttt{ring}}$) and obtains $\mathsf{compk}', \pi', \mathsf{b}' = 0$. Then, it runs ZKCont.Reprove($crs, X', \pi', \mathsf{b}', \mathcal{R}^{\texttt{inner}}_{\texttt{ring}}$) and obtains $(\mathsf{compk}, \pi_1, \mathsf{b})$. Finally, it lets $\mathsf{preout} = \mathsf{sk} H_{\mathbf{G}}(\mathsf{input})$ and runs $\mathsf{NIZK}_{\mathcal{R}_{eval}}.\mathsf{Prove}(\mathsf{compk}, \mathsf{preout}, \mathsf{input}; \mathsf{sk}, \mathsf{b}, \mathsf{input}) \to \pi_2$ as described in Section 5 with $\mathsf{ad}' = \mathsf{ad} \mathbin{+\!\!+} \pi_{\texttt{ring}} \mathbin{+\!\!+} \mathsf{comring}$. In the end, it returns the ring signature $\sigma = (\mathsf{preout}, \pi_1, \pi_2, \mathsf{compk})$.
- rVRF.Verify : $(\mathsf{comring}, \mathsf{input}, \mathsf{ad}, \sigma) \mapsto (1, \mathsf{out}) \vee (0, \perp)$ it parses σ as $\mathsf{preout}, \pi_1, \pi_2, \mathsf{compk}$ and runs $\mathsf{NIZK}_{\mathcal{R}_{\texttt{rvrf}}}.\mathsf{Verify}(\mathsf{comring}, \mathsf{preout}, \mathsf{input}, \mathsf{out}; \pi_{rvrf})$ i.e., runs ZKCont.Verify(crs, comring, compk, $\pi_1, \mathcal{R}^{\texttt{inner}}_{\texttt{ring}}$) and $\mathsf{NIZK}_{\mathcal{R}_{eval}}.\mathsf{Verify}((\mathsf{compk}, (\mathsf{preout}, \mathsf{input}, \mathsf{out})), \pi_2)$. If all verify, it outputs $(1, \mathsf{out} = H(\mathsf{input}, \mathsf{preout}))$. Otherwise, it returns $(0, \perp)$.

Theorem 3. *Our specialized* rVRF *over pp_{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 7 and $\mathsf{NIZK}_{\mathcal{R}_{eval}}$ is zero-knowledge and knowledge sound, the decisional Diffie-Hellman (DDH) problem are hard in $\mathbf{G}$ (so the CDH problem is hard as well) and the commitment scheme* Com *is binding and perfectly hiding.*

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

```
1: $\mathsf{comring}, \mathsf{opring} \leftarrow \mathsf{rVRF.CommitRing}(\mathsf{ring}, \mathsf{pk})$
2: $\mathsf{compk}', \pi', \mathsf{b}' \leftarrow \mathsf{ZKCont.Preprove}(crs, \mathsf{comring}, \mathsf{sk}, (r, \mathsf{opring}), \mathcal{R}^{\texttt{inner}}_{\texttt{ring}})$
3: $\mathsf{compk}, \pi_1, \mathsf{b} \leftarrow \mathsf{ZKContReprove}(crs, \mathsf{compk}', \pi', \mathsf{b}', \mathcal{R}^{\texttt{inner}}_{\texttt{ring}})$
4: $c, s_1, s_2 \leftarrow\!\!\$\ \mathbb{F}_p$
5: $\pi_{eval} \leftarrow (c, s_1, s_2)$
6: return $\sigma = (\pi_{eval}, \pi_{ring}, \mathsf{compk}, \mathsf{comring}, W)$
```

Proof Sketch: The security proof follows the same proof we have for Theorem 6 except the parts that we run extractor and simulator algorithms in $\mathsf{NIZK}_{\mathcal{R}_{\texttt{ring}}}$ and $\mathsf{NIZK}_{\mathcal{R}_{eval}}$. We construct the same Sim described in the proof of Theorem 6 and use the result of Lemma 4.The only slight difference is in Lemma 6 since Gen_{sign} is different than Algorithm 1. There, we run the simulator and extractor of $\mathsf{NIZK}_{\mathcal{R}_{\texttt{rvrf}}}$. See Appendix A for more details.

Instantiation with $\mathsf{SpecialG}$*:* Since $\mathsf{SpecialG}$ is ZKCont, we can instantiate our protocol with $\mathsf{SpecialG}$. In this case, we present an appropriate $\mathsf{Com.Commit(sk)}$ algorithm that together with $\mathsf{SpecialG}$ efficiently instantiate the NIZK for $\mathcal{R}^{\mathtt{inner}}_{\mathtt{ring}}$. To make this efficiently provable inside the SNARK, we use the Jubjub 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 $\mathbf{G}$ used in our ring VRF construction[5]. 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.

Efficiency with $\mathsf{SpecialG}$*:* If we deploy a $\mathsf{SpecialG}$ for $\mathcal{R}^{\mathtt{inner}}_{\mathtt{ring}}$ to generate a ring VRF signature for the same ring and for a different input, we need to run $\mathsf{SpecialG.Reprove}$ and $\mathsf{NIZK}_{\mathcal{R}_{eval}}.\mathsf{Prove}$. $\mathsf{NIZK}_{\mathcal{R}_{eval}}.\mathsf{Prove}$ requires only three scalar multiplications and $\mathsf{SpecialG.Reprove}$ requires five: two in G_2 and 3 in G_1.

7 Ring updates

We now discuss the performance of $\pi_{\mathtt{fast}}$. Although our $\mathsf{rVRF.Sign}$ runs fast, all users should update their stored zkSNARK $\pi_{\mathtt{fast}}$ every time ring changes, but zero knowledge continuations help here too.

7.1 Merkle trees

Our $\mathsf{rVRF}.\{\mathsf{CommitRing}, \mathsf{OpenRing}\}$ could implement a Merkle tree using a zkSNARK friendly hash function like Poseidon [23], giving $O(\log|\mathsf{ring}|)$ prover time. At least one Poseidon [23] provides arity four with only 600 R1CS constraints. We need roughly 700 R1CS constraints for each fixed based scalar multiplication too, so the flavor of $\pi_{\mathtt{fast}}$ costs under 12k R1CS constraints for a ring with four billion people.

7.2 Side channels

In $\pi_{\mathtt{fast}}$, one might dislike processing secret key material inside the Groth16 prover for $\pi_{\mathtt{fast}}$. Adversaries could trigger $\pi_{\mathtt{fast}}$ recomputation only by updating the ring, but this still presents a side channel risk.

If concerned, one could address this via a second zk continuation that splits $\pi_{\mathtt{fast}}$ into a Groth16 π_{sk} and a Groth16 or KZG π_{pk} for two respective languages:

$$\mathcal{L}^{\mathtt{inner}}_{\mathsf{pk}} = \{\, J_{\mathsf{pk}}, \mathsf{comring} \mid \exists \mathsf{opring} \text{ s.t. } J_{\mathsf{pk}} = \mathsf{rVRF.OpenRing}(\mathsf{comring}, \mathsf{opring}) \,\}, \quad \text{and}$$

$$\mathcal{L}^{\mathtt{inner}}_{\mathsf{sk}} = \left\{ \mathsf{sk}_0 + \mathsf{sk}_1 2^{128}, J_{\mathsf{pk}} \;\middle|\; \exists d \text{ s.t. } J_{\mathsf{pk}} = \mathsf{sk}_0 J_0 + \mathsf{sk}_1 J_1 + dJ_2 \right\}.$$

We now prove π_{sk} only once *ever* during secret key generation, which largely eliminates any side channel risks. We do ask verifiers compute more pairings, but nobody cares when the VRF verifiers are few in number or institutional, as in many applications. We also ask provers rerandomize both π_{sk} and π_{pk}, but this costs relatively little. Assuming π_{pk} is Groth16 then we need a proof-of-knowledge for the desired structure of J_{pk} too. All totaled this almost doubles the size and complexity of our ring VRF signature.

There is no "arrow of time" among zk continuations per se, but as π_{sk} bridges between the PedVRF and π_{pk}, one might consider the π_{sk}-to-π_{pk} continuation to be "time reversed", in that the "middle" continuation is proved first.

[5] This condition can be satisfied if $\mathbb{J}$ is an Edwards curve with a cofactor.

7.3 Polynomial commitments

As π_{pk} became rather simple, there exists an alternative formulation: $\mathsf{comring}$ could be a KZG polynomial commitment [27] to users' J_{pk}s, while π_{pk} itself becomes an opening at a secret location, like Caulk+ [39] or Caulk [42]. We benefit from faster ring updates this way, but pay in increased verifier time and increased marginal prover time.

7.4 Append only rings

As a slight variation, we could build ring using append only structures like some blockchains, in which case we should split $\mathsf{rVRF.OpenRing}$ differently between an inner ring block or epoch proof $\mathcal{L}_{\mathtt{block}}$, which we only prove once like π_{sk} above, and a chain state proof $\mathcal{L}_{\mathtt{chain}}$, which extends this inner ring to the growing blockchain. Now our inner SNARKs pass a `blk` parameter, which our zero-knowledge continuation transforms into a opaque commitment `comblk`, thereby requiring a proof-of-knowledge.

$$\mathcal{L}^{\mathtt{inner}}_{\mathtt{chain}} = \left\{ \mathtt{blk}, \mathtt{chain} \mid \mathtt{blk} \in \mathtt{chain} \right\}, \quad \text{and}$$

$$\mathcal{L}^{\mathtt{inner}}_{\mathtt{block}} = \left\{ \mathsf{sk}_0 + \mathsf{sk}_1 2^{128}, \mathtt{blk} \;\middle|\; \exists d, \mathsf{opring} \text{ s.t. } \begin{array}{c} \mathsf{OpenRing}(\mathtt{blk}, \mathsf{opring}) \\ = \mathsf{sk}_0 J_0 + \mathsf{sk}_1 J_1 + d J_2 \end{array} \right\}.$$

We suggest appending `blk` to a polynomial commitment using [41], which then $\mathcal{L}_{\mathtt{chain}}$ blind opens via Caulk+ [39] as above.

7.5 Expiration and revocation

We expect expiration and revocation would be required for append only rings like blockchains, or say a zero-knowledge proof of a certificate.

For expiry, we suggest π_{sk} or $\mathcal{L}_{\mathtt{block}}$ commit to the expiration date alongside the secret key in their X, and then π_{pk} or $\mathcal{L}_{\mathtt{chain}}$ enforce expiration, but really even PedVRF could enforce expiration.

A revocation list could be enforced by a non-membership proof in π_{pk} or $\mathcal{L}_{\mathtt{chain}}$. We expect a revocation list updates only rarely compared with ring itself though, which makes doing this non-membership proof inside some separate zero-knowledge continuation tempting too. A deployment faces should make this choice carefully.

8 Anonymized ring unions

We briefly discuss ring VRFs whose ring consists of the union of several smaller rings, but which hide to which ring the user belongs. In this, we bring out one interesting zero-knowledge continuation technique.

8.1 Identical circuit

As a first step, if all rings use the same circuit, then we hide the ring among several rings using a second zero-knowledge continuation, not unlike §7.2. We could then blind open a polynomial commitment [27] to our $\mathsf{comring}$ choices, Caulk+ [39] or Caulk [42] or similar as in §7.3.

As a special case, if users cannot change their keys too quickly, then one could reduce the frequency with which users reprove their original zero-knowledge by using multiple $\mathsf{comring}$ choices across the history of the same evolving ring database.

8.2 Multi-circuit

We need a new trick if the χ_i come from different circuit's trusted setups. A priori, our zero-knowledge continuation $\pi_{\mathtt{fast}}$ fixes some $G = \chi_1$, which reveals the circuit, due to its dependence upon the SRS like

$$\chi_1 = \frac{\beta u_1(\tau) + \alpha v_1(\tau) + w_1(\tau)}{\gamma} \cdot \mathsf{g}_1 \ .$$

Instead, we propose to stabilize the public input SRS elements across circuits: We choose $\chi_{1,\gamma}$ independently before selecting the circuit or running its trusted setup. We then merely add an SRS element $\chi_{1,\delta}$, for usage in C, that binds our independent $\chi_{1,\gamma}$ to the desired definition, so

$$\chi_{1,\delta} := \frac{\beta u_1(\tau) + \alpha v_1(\tau) + w_1(\tau) - \gamma \chi_{1,\gamma}}{\delta} \cdot \mathsf{g}_1.$$

At this point, we replace χ_1 by $\chi_{1,\gamma}$ everywhere and our proofs add $\mathsf{comring}\,\chi_{1,\delta}$ to C.

In this way, all ring membership circuits could share identical public input SRS points $\chi_{1,\gamma}$, and similarly χ_0 if desired.

At this point, one still needs to hide the SRS elements $\delta \cdot \mathsf{g}_2, \gamma \cdot \mathsf{g}_2 \in \mathbf{G}_2$ and $e(\alpha \cdot \mathsf{g}_1, \beta \cdot \mathsf{g}_2) \in \mathbf{G}_T$. We leave this as an exercise to the reader.

9 Application: Identity

Ring VRFs yield anonymous identity systems: After a user and service establish a secure channel and the server authenticates itself with certificates, then the user authenticates themselves by providing an anonymous VRF signature with input input being the service's identity, thus creating an pseudonymous identified session with a pseudonym unlinkable from other contexts.

We expand this identified session workflow with an extra update operation suitable for our ring VRF's amortized prover. We discuss only $\pi_{\mathtt{fast}}$ here but all techniques apply to π_{sk} and π_{pk} similarly.

- *Register* – Adds users' public key commitments into some ring, after verifying the user does not currently exist in ring.
- *Update* – User agents regenerate their stored SNARK $(\mathsf{pk}, \pi^{\mathtt{inner}}_{\mathtt{fast}})$ using $\mathsf{SpecialG.Preprove}((\mathsf{sk}_1, \mathsf{sk}_2, \mathsf{opring}); (\mathsf{sk}, \mathsf{comring}))$ each time ring changes, perhaps even receiving $\mathsf{comring}$ and opring from some ring management service.
- *Identify* – Our user agent first opens a standard TLS connection to a server input, both checking the server's name is input and checking certificate transparency logs, and then computes the shared session id ad. Our user agent computes the user's identity $\mathtt{id} = \mathsf{PedVRF.Eval}(\mathsf{sk}, \mathsf{input})$ on the server id input, Our user agent next rerandomizes $\pi_{\mathtt{fast}}$, compk, and b using $\mathsf{SpecialG.Reprove}(\mathsf{pk}, \pi^{\mathtt{inner}}_{\mathtt{fast}})$, computes $\sigma = \mathsf{PedVRF.Sign}(\mathsf{sk}, \mathsf{b}, \mathsf{input}, \mathsf{ad} \mathbin{+\!\!+} \mathsf{compk} \mathbin{+\!\!+} \pi_{\mathtt{fast}})$, and finally sends the server their ring VRF signature $(\mathsf{compk}, \pi_{\mathtt{fast}}, \sigma)$
- *Verify* – After receiving $(\mathsf{compk}, \pi_{\mathtt{fast}}, \sigma)$ in channel ad, the server named input checks $\mathsf{SpecialG.Verify}(\mathsf{comring}, (\mathsf{compk}, \pi_{\mathtt{fast}}))$, checks the VRF signature, and obtains the user's identity $\mathtt{id}$, ala
  $\mathtt{id} = \mathsf{PedVRF.Verify}(\mathsf{compk}, \mathsf{input}, \mathsf{ad} \mathbin{+\!\!+} \mathsf{compk} \mathbin{+\!\!+} \pi_{\mathtt{fast}}, \sigma)$.

9.1 Browsers

We must not link users' identities at different web sites, so user agents should carefully limit cross site resource loading, referrer information, etc. User agents could always load purely static resources, without metadata like cookies or referrer information. At least Tor browser already takes cross site resource concerns seriously, while Safari and Brave may limit invasive cross site resources too.

We somewhat trust the CAs and CT log system with users' identities in the above protocol, in that users could login to a site with fraudulent credentials. We think cross site restrictions limit this attack vector. If stronger defenses are desired then instead of input being the site name, input could be a public "root" key for the specific site, which then also certifies its TLS certificate. Ideally its secret key remains air gaped.

9.2 AML/KYC

We shall not discuss AML/KYC in detail, because the entire field lacks clear goals, and thus winds up being ineffective [38]. We do however observe that AML/KYC typically conflicts with security and privacy laws like GDPR. As a compromise between these regulations, one needs a compliance party who know users' identities, while another separate service party knows the users' activities. We propose a safer and more efficient solution:

Instead our compliance party becomes an identity issuer who maintains a public ring, and privately knows the users behind each public key. As above, identity systems could employ ring freely for diverse purposes. If later asked or subpoenaed, users could prove their relevant identities to investigators, or maybe prove which services they use and do not use.

Interestingly PedVRF could run "backwards" like $H_{\mathbf{G}'}(\mathsf{input}) \neq \mathsf{sk}^{-1}\,\mathsf{preout}$ to show a ring VRF output associated to preout does not belong to the user, without revealing the users' identity $H'(\mathsf{input}, \mathsf{sk}\, H_{\mathbf{G}'}(\mathsf{input}))$ to investigators.

Our applications mostly ignore key multiplicity. AML/KYC demands suspects prove non-involvement using ring VRFs.

Definition 10. *We say* rVRF *is* exculpatory *if we have an efficient algorithm for equivalence of public keys, but a PPT adversary $\mathcal{A}$ cannot find non-equivalent public keys $\mathsf{pk}_0, \mathsf{pk}_1$ with colliding VRF outputs.*

A priori, our JubJub representations $\mathsf{sk}_0 J_0 + \mathsf{sk}_1 J_1$ used in §6.3 and §7.2 costs us exculpability from Definition 10.

There is however a natural *exculpable public key* flavor (pk, σ), in which $\sigma = \mathsf{Sign}(\mathsf{sk}, \mathsf{CommitRing}(\{\mathsf{pk}\}, \mathsf{pk}).\mathsf{opring}, \texttt{ring\_name}, "")$. The singleton ring $\{\mathsf{pk}\}$ ensure that $\mathsf{Verify}(\mathsf{CommitRing}(\{\mathsf{pk}\}), \texttt{ring\_name}, "", \sigma)$ uniquely determines the secret key, so exculpability holds if joining the ring requires (pk, σ).

9.3 Moderation

All discussion or collaboration sites have behavioral guidelines and moderation rules that deeply impact their culture and collective values.

Our ring VRFs enables a simple blacklisting operation: If a user misbehaves, then sites could blacklist or otherwise penalizes their site local identity `id`. As `id` remains unlinked from other sites, we avoid thorny questions about how such penalties impact the user elsewhere, and thus can assess and dispense justice more precisely.

At the same time, there exist sites who must forget users' histories eventually, like under some "right to be forgotten" principle, either GDPR compliance or an ethical principle of social mistakes being ephemeral.

We obtain ephemeral identities if input consists of the site name plus the current year and month, or some other approximate date. In this way, users have only one stable `id` within the approximate date range, but they obtain fresh `id`s merely by waiting until the next month.

We could adjust PedVRF to simultaneously prove multiple VRF input-output pairs $(\mathsf{input}_j, \mathtt{id}_j)$. As in [18], we merely delinearize inbase and preout in Sign and Verify like:

$$x = H(\mathsf{input}_j, \mathtt{id}_j, \ldots, \mathsf{input}_j, \mathtt{id}_j)$$

$$\mathsf{inbase} = \sum_j H_p(x, j)\, \mathsf{inbase}_j$$

$$\mathsf{preout} = \sum_j H_p(x, j)\, \mathsf{preout}_j$$

As doing so links these pairs together, we could link together two or more ephemeral identities like this to obtain a semi-permanent identity with user controlled revocation: As login, our site demands two linked input-output pairs given by $\mathsf{input}_1 = \texttt{site\_name} \mathbin{+\!\!+} \texttt{current\_month}$ and $\mathsf{input}_2 = \texttt{site\_name} \mathbin{+\!\!+} \texttt{registration\_month}$, so users could have multiple active pseudo-nyms given by $\mathtt{id}_2$, but only one active pseudo-nym per month, enforced by deduplicating $\mathtt{id}_1$, which still prevents spam and abuse.

If instead our site associates pseudo-nyms to their most recently seen $\mathtt{id}_1$, then we could link adjacent months, meaning input_j is defined by the jth previous month, until reaching a previously used $\mathtt{id}_1$. In this model, pseudo-nyms could be abandoned and replaced, but abandoned pseudo-nyms cannot then be reclaimed without linking intervening dates. Although more costly, sites could permanently bans a few problematic users via the inequality proofs described in §9.2 too.

In these ways, sites encode important aspects of their moderation rules into the ring VRF inputs they demand.

9.4 Reduced pairings

At a high level, we distinguish moderation-like applications discussed above, which resemble classic identity applications like AML/KYC, from rate limiting applications discussed in the next section. In moderation-like applications, ring VRF outputs become long-term stable identities, so users typically reidentify themselves many times to the same sites, reusing the exact same input.

As an optimization, our zero-knowledge continuation could reuse the same compk and $\pi_{\mathtt{fast}}$ for the same input, so that verifiers could memoize their verifications of $\pi_{\mathtt{fast}}$. We spend most verifier time checking the Groth16 pairing equation, so this saves considerable CPU time.

As a concrete example, our coefficients r_1, r_2, b used for rerandomization in §6 could be chosen deterministically like $r_1, r_2, b \leftarrow H(\mathsf{sk}, \mathsf{input})$. In this way, each (helpful) user's $\mathtt{id}$ has a unique $\pi_{\mathtt{fast}}$, which verifiers could memoize by storing $(\mathtt{id}, H(\mathsf{compk} + \!\!+\, \pi_{\mathtt{fast}}), \mathtt{dates})$ after their first verification, but then skipping the Groth16 check after merely rechecking the hash $H(\mathsf{compk} +\!\!+\, \pi_{\mathtt{fast}})$.

We could risk denial-of-service attacks by users who vary r_1, r_2, b randomly however. We therefore suggest $\mathtt{dates}$ record the last several previous dates when $H(\mathsf{compk} +\!\!+\, \pi_{\mathtt{fast}})$ changed. We rate limit or verify more lazily users with many nearby login dates

10 Application: Rate limiting

We showed in §9 how ring VRFs give users only one unique identity for each input input. We explained in §9.3 that choosing input to be the concatenation of a base domain and a date gives users a stream of changing identities. We next discuss giving users exactly $n > 1$ ring VRF outputs aka "identities" per date, as opposed to one unique identity

As a trivial implementation, we could include a counter $k = 1 \ldots n$ in input, so $\mathsf{input} = \mathtt{domain} +\!\!+ \mathtt{date} +\!\!+ k$.

10.1 Avoiding linkage

Our trivial implementation leaks information about ring VRF outputs' ownership by revealing k: An adversary Eve observes two ring VRF signatures with the same $\mathtt{domain}$ and $\mathtt{date}$ so $\mathsf{input}_i = \mathtt{domain} +\!\!+ \mathtt{date} +\!\!+ k_i$ for $i = 1, 2$, but with different outputs out_1 and out_2. If $k_1 \neq k_2$ then Eve learns nothing, but if $k_1 = k_2$ then Eve learns that $sk_1 \neq \mathsf{sk}_2$, maybe representing different users.

We do not necessarily always care if Eve learns this much information, but scenarios exist in which one cares. We therefore briefly describe several mitigation:

If n remains fixed forever, then we could simply let all users register n ring VRF public keys in ring. If n fluctuates under an upper bound N, then we could create N rings ring_i for $i = 1 \ldots N$, and then blind $\mathsf{comring}$ in $\pi_{\mathtt{fast}}$ similarly to §8.

Although simple, these two approaches require users construct n or N different π_{pk} proofs every time ring updates.

Instead of proving ring membership of one public key, π_{pk} could prove ring membership of a Merkle commitment to multiple keys, so users have $\pi^1_{\mathsf{sk}}, \ldots, \pi^N_{\mathsf{sk}}$ for each of their multiple keys.

In principle, there exists ring VRFs that hide parts of their input input, but still fit our abstract formulation in §2. Although interesting, we caution these bring performance concerns not discussed here, so deployments should consider if leaking k suffices.

10.2 Ration cards

As a species, we expect $+3°$C over the pre-industrial climate by 2100 [1], or more likely above $+4°$C given tipping points [33]. At these levels, we experience devastating famines as the Earth's carrying capacity drops below one billion people [40]. In the near term, our shortages of resources, energy, goods, water, and food shall steadily worsen over the next several decades, due to climate change, ecosystem damage or collapse, and resource exhaustion ala peak oil. We expect synchronous crop failures around the 2040s in particular [15]. Invariably, nations manage shortages through rationing, like during WWI, WWII, and the oil shocks.

Ring VRFs support anonymous rationing: Instead of treating ring VRF outputs like identities, we treat them like nullifiers which could each be spent exactly once.

We fix a set U of limited resource types, overseen by an authority who certifies verifiers from a key $\texttt{root}$. We dynamically define an expiry date e_{u,d_0} and an availability n_{u,d_0}, both dependent upon the resource $u \in U$ and current date d_0. We typically want a randomness beacon r_d too, which prevents anyone learning r_d much before date d. As ring VRF inputs, we choose $\mathsf{input} = \texttt{root} \mathbin{+\!\!+} u \mathbin{+\!\!+} r_d \mathbin{+\!\!+} d \mathbin{+\!\!+} k$ where $u \in U$ denotes a limited resource, d denotes an non-expired date meaning $e_{u,d_0} < d \leq d_0$, and $1 \leq k \leq n_{u,d_0}$. In this way, our rationing system controls both daily consumption via n_{u,d_0} and time shifted demand via expiry time e_{u,d_0}.

Importantly, our rationing system retains ring VRF outputs as nullifiers, filed under their associated date d and resource u, so nullifiers expire once $d \leq e_{u,d_0}$ which permits purging old data rapidly.

We remark that fully transferable assets could have constrained lifetimes too, which similarly eases nullifier management when implements using blind signatures, ZCash sapling, etc. Yet, all these tokens require an explicit issuance stage, while ring VRFs self-issue.

Among the political hurdles to rationing, we know certificates have a considerable forgery problem, as witnessed by the long history of fraudulent covid and TLS certificates. It follows citizens would justifiably protest to ration carts that operate by simple certificates. Ring VRFs avoid this political unrest by proving membership in a public list.

10.3 Multi-constraint rationing

As in §9.3, we could impose simultaneous rationing constraints for multiple resources $u_1, \ldots, u_k$ by producing one ring VRF signature in which PedVRF proves correctness of pre-outputs for multiple messages $\mathsf{input}_j = \texttt{root} \mathbin{+\!\!+} u_j \mathbin{+\!\!+} r_d \mathbin{+\!\!+} d \mathbin{+\!\!+} k$ for $j = 1 \ldots k$.

As an example, purchasing some prepared food product could require spending rations for multiple base food sources, like making a cake from wheat, butter, eggs, and sugar.

10.4 Decommodification

There exist many reasons to decommodify important services, like energy, water, or internet, beyond rationing real physical shortages. Ring VRFs fit these cases using similar input formulations.

As an example, a municipal ISP allocates some limited bandwidth capacity among all residents. It allocates bandwidth fairly by verifying ring VRFs signatures on hourly input and then tracking nullifiers until expiry.

Aside from essential government services, commercial service providers typically offers some free service tier, usually because doing so familiarizes users with their intimidating technical product.

Some free and paid tier examples include DuoLingo's hearts on mobile, continuous integration testing services, and many dating sites.

A priori, rate limiting cases benefit from unlinkability among individual usages, not merely at some site boundary like moderation requires. We thus use each ring VRF output only once, which prevents our cashing trick of §9.4 from reducing verifier pairings.

Although rationing sounds valuable enough, we foresee services like ISP, VPNs, or mixnets having many low value transactions. In such cases, ring VRFs could authorize issuing a limited number of fast simple single-use blind issued credentials, like blind signatures ala GNU Taler [8] or PrivacyPass OPRF tokens [18], which both solve the leakage of k above too. In principle, commercial service providers could sell the same tokens, which avoids leaking whether the user uses the free or commercial tier.

10.5 Delegation

Almost all single-use blind signed tokens have an implicit delegation protocol, in which token holders transfer token credentials without sacrificing their own access. As double spending remains possible, delegatees must trust delegators. GNU Taler [8] argues against taxing such trusting transfers, like when parents give their kids spending money, but enforces taxability only when also preventing double spending.

In our rationing scheme, spenders authenticate their specific spending operations inside the associated data ad in a rVRF-AD signature. As doing so requires knowing sk, delegators place enormous trust in delegatees, which likely precludes say parents delegating to children.

We could however achieve delegation by treating the ring VRF like a certificate that authenticates another public key held by the delegatee. In fact, delegators could limit delegatees uses too in this certificate, like how GNU Taler achieves parental restrictions.

We remark that PedVRF has adaptor signatures aka implicit certificate mode: A delegatee learns the full ring VRF signature, but the delegatee hides the blinding factor signature s_1 in PedVRF from downstream recipients, and instead merely prove knowledge of s_1, say via a key exchange or another Schnorr signature with the base point K. EC VRFs lack this mode.

References

1. Climate change 2022: Impacts, adaptation and vulnerability. working group ii contribution to the ipcc sixth assessment report. Cambridge University Press. In Press.
2. Karim Baghery, Markulf Kohlweiss, Janno Siim, and Mikhail Volkhov. Another look at extraction and randomization of groth's zk-snark. In Nikita Borisov and Claudia Diaz, editors, *Financial Cryptography and Data Security*, pages 457–475, Berlin, Heidelberg, 2021. Springer Berlin Heidelberg. `https://ia.cr/2020/811`.
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. Jeffrey Burdges, Handan Kılınç Alper, Alistair Stewart, and Sergey Vasilyev. Sassafras and semi-anonymous single leader election. Cryptology ePrint Archive, Paper 2023/031, 2023. `https://eprint.iacr.org/2023/031`.
8. Jeffrey Burdges, Florian Dold, and Christian Grothoff. Robust and income-transparent online e-cash.
9. Privacy by Design Foundation. Irma docs v0.2.0. `https://irma.app/docs/v0.2.0/overview/`.
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. Chatham House. Climate change risk assessment 2021.
16. David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology*, pages 199–203, Boston, MA, 1983. Springer US.

17. 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.
18. Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *Proceedings on Privacy Enhancing Technologies*, 2018:164–180, 06 2018.
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. 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`.
21. 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`.
22. 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.
23. Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. Cryptology ePrint Archive, Paper 2019/458, 2019. `https://eprint.iacr.org/2019/458`.
24. 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.
25. Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *Journal of the ACM (JACM)*, 59(3):1–35, 2012.
26. Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. ZCash protocol specification. `https://zips.z.cash/protocol/protocol.pdf`. Version 2022.3.8 [NU5], 15 September 2022.
27. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, pages 177–194, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
28. Joe Kilian. Uses of randomness in algorithms and protocols. PhD Thesis. Massachusetts Institute of Technology, 1990.
29. 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.
30. 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.
31. 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.
32. Mark Manulis, Nils Fleischhacker, Felix Günther, Franziskus Kiefer, and Bertram Poettering. Group signatures: Authentication with privacy. BSI, 2011. `https://www.franziskuskiefer.de/paper/GruPA.pdf` and `https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/GruPA/GruPA.pdf`.
33. David I. Armstrong McKay, Arie Staal, Jesse F. Abrams, Ricarda Winkelmann, Boris Sakschewski, Sina Loriani, Ingo Fetzer, Sarah E. Cornell, Johan Rockström, and Timothy M. Lenton. Exceeding 1.5°c global warming could trigger multiple climate tipping points. *Science*, 377(6611):286–295, Sep 2022.
34. 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.
35. Nate Otto, Sunny Lee, Brian Sletten, Daniel Burnett, Manu Sporny, and Ken Ebert. Verifiable credentials use cases. W3C Working Group Note 24 September 2019. `https://www.w3.org/TR/vc-use-cases/`.
36. 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`.
37. Trevor Perrin. The xeddsa and vxeddsa signature schemes. Revision 1, 2016-10-20. `https://signal.org/docs/specifications/xeddsa/`.
38. Ronald F. Pol. Anti-money laundering: The world's least effective policy experiment? together, we can fix it. *Policy Design and Practice*, 3(1):73–94, 2020.
39. Jim Posen and Assimakis A. Kattis. Caulk+: Table-independent lookup arguments. Cryptology ePrint Archive, Paper 2022/957, 2022. `https://eprint.iacr.org/2022/957`.

40. David Spratt. At 4°c of warming, would a billion people survive? what scientists say.
41. Alin Tomescu, Ittai Abraham, Vitalik Buterin, Justin Drake, Dankrad Feist, and Dmitry Khovratovich. Aggregatable subvector commitments for stateless cryptocurrencies. Cryptology ePrint Archive, Report 2020/527, 2020. `https://ia.cr/2020/527`.
42. Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. Cryptology ePrint Archive, Paper 2022/621, 2022. `https://eprint.iacr.org/2022/621`.

A Security of Our Ring VRF Construction

A.1 Security of Our Protocol in Section 5

Before we start to analyse our protocol, we should define the algorithm Gen_{sign} for $\mathcal{F}_{\mathsf{rvrf}}$ and show that $\mathcal{F}_{\mathsf{rvrf}}$ with Gen_{sign} satisfies the anonymity defined in Definition 6. $\mathcal{F}_{\mathsf{rvrf}}$ that rVRF realizes runs Algorithm 1 to generate honest signatures.

Lemma 4. *$\mathcal{F}_{\mathsf{rvrf}}$ running Algorithm 1 satisfies anonymity defined in Definition 6 assuming that $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{ring}}}$ is a zero-knowledge and Pedersen commitment is perfectly hiding.*

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 let $\pi_{ring} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{\mathsf{ring}}}.\mathsf{Simulate}(G, K, \mathbf{G}, \mathsf{comring}, \mathsf{compk})$. Since $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{ring}}}$ is zero knowledge, there exists an algorithm $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{ring}}}.\mathsf{Simulate}$ which generates a proof which is indistinguishable from the proof generated from $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{ring}}}.\mathsf{Prove}$. Therefore, our reduced game is indistinguishable from the anonymity game. Since in this game, no public 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 next show that rVRF realizes $\mathcal{F}_{\mathsf{rvrf}}$ in the random oracle model under the assumption of the hardness of the decisional Diffie Hellman (DDH).

Theorem 4. rVRF *over pp_{rvrf} realizes $\mathcal{F}_{\mathsf{rvrf}}$ running Gen_{sign} in Algorithm 1 [11,12] in the random oracle model assuming that $\mathsf{NIZK}_{\mathcal{R}_{eval}}$ and $\mathsf{NIZK}_{\mathcal{R}_{ring}}$ are zero-knowledge and knowledge sound, the decisional Diffie-Hellman (DDH) problem are hard in $\mathbf{G}$ (so the CDH problem is hard as well) and the commitment scheme* Com *is binding and perfectly hiding.*

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 obtains a secret and public key pair $x = (\mathsf{sk}, r)$ and pk by running $\mathsf{rVRF.KeyGen}$. It adds pk to lists `honest_keys` and `signing_keys` as a key of P_i. In the end, Sim returns $(\mathsf{verificationkey}, \mathsf{sid}, x, \mathsf{pk})$ to $\mathcal{F}_{\mathsf{rvrf}}$. Sim lets $\mathsf{public\_keys}[X] = \mathsf{pk}$ and $\mathsf{secret\_keys}[X] = (\mathsf{sk}, r)$ where $X = \mathsf{sk}G$.
- **[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 `honest_keys` which is stored as a key of P_i and adds pk to `malicious_keys`.
- **[Simulation of the random oracles:]** We describe how Sim simulates the random oracles $H_{\mathbf{G}}, H, H_p$ against the real world adversaries.
  Sim simulates the random oracle $H_{\mathbf{G}}$ as described in Figure 2. It selects a random element h from $\mathbb{F}_p$ for each new input and outputs hG as an output of the random oracle $H_{\mathbf{G}}$. Thus, Sim knows *the discrete logarithm of each random oracle output of* $H_{\mathbf{G}}$.
  The simulation of the random oracle H is less straightforward (See Figure 3). The value W can be a pre-output of an input of a malicious party or can be an anonymous key of m generated by $\mathcal{F}_{\mathsf{rvrf}}$ for an honest party. Sim does not need to know about this at this point but H should output `evaluations`$[m, W]$ in both cases to be consistent with $\mathcal{F}_{\mathsf{rvrf}}$. Sim considers W as if it is a pre-output. So, Sim first obtains

$\mathcal{F}_{\mathsf{rvrf}}$ 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 a party P_i, send $(\mathsf{keygen}, \mathsf{sid}, \mathsf{P}_i)$ to the simulator Sim. Upon receiving a message $(\mathsf{verificationkey}, \mathsf{sid}, x, \mathsf{pk})$ from Sim, verify that x, pk has not been recorded before for sid i.e., there exists no (x', pk') in $\mathtt{signing\_keys}$ such that $x' = x$ or $\mathsf{pk}' = \mathsf{pk}$. If it is the case, store in the table $\mathtt{signing\_keys}$, under P_i, the value x, pk and return $(\mathsf{verificationkey}, \mathsf{sid}, \mathsf{pk})$ to P_i.

[**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 $(\mathsf{corrupted}, \mathsf{sid}, \mathsf{P}_i)$.

[**Malicious Ring VRF Evaluation.**] upon receiving a message $(\mathsf{eval}, \mathsf{sid}, \mathsf{pk}_i, W, \mathsf{input})$ from Sim, if pk_i is recorded under an honest party's identity or if there exists $W' \neq W$ where $\mathtt{anonymous\_key\_map}[\mathsf{input}, W'] = \mathsf{pk}_i$, 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 $\mathtt{anonymous\_key\_map}[\mathsf{input}, W]$ is not defined before, set $\mathtt{anonymous\_key\_map}[\mathsf{input}, W] = \mathsf{pk}_i$ and let $y \leftarrow\$ \mathcal{S}_{eval}$ and set $\mathtt{evaluations}[\mathsf{input}, W] = y$.

In any case (except ignoring), obtain $y = \mathtt{evaluations}[\mathsf{input}, W]$ and return $(\mathsf{evaluated}, \mathsf{sid}, \mathsf{input}, \mathsf{pk}_i, W, y)$ to P_i.

[**Honest Ring VRF Signature and Evaluation.**] upon receiving a message $(\mathsf{sign}, \mathsf{sid}, \mathsf{ring}, \mathsf{pk}_i, \mathsf{ad}, \mathsf{input})$ 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 $\mathtt{anonymous\_key\_map}[\mathsf{input}, W'] = \mathsf{pk}_i$, let $W \leftarrow\$ \mathcal{S}_W$ and let $y \leftarrow\$ \mathcal{S}_{eval}$. Set $\mathtt{anonymous\_key\_map}[\mathsf{input}, W] = \mathsf{pk}_i$ and set $\mathtt{evaluations}[\mathsf{input}, W] = y$.

In any case (except ignoring), obtain W, y where $\mathtt{anonymous\_key\_map}[\mathsf{input}, W] = \mathsf{pk}_i$, $\mathtt{evaluations}[\mathsf{input}, W] = y$ and (x, pk) is in $\mathtt{signing\_keys}$. Then run $\mathsf{Gen}_{sign}(\mathsf{ring}, W, x, \mathsf{pk}, \mathsf{ad}, \mathsf{input}) \rightarrow \sigma$. Record $[\mathsf{input}, \mathsf{ad}, W, \mathsf{ring}, \sigma, 1]$. Return $(\mathsf{signature}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, \mathsf{input}, y, \sigma)$ to P_i.

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

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

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

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

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

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

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

– if $W \notin \mathcal{W}[\mathsf{input}, \mathsf{ring}]$, add W to $\mathcal{W}[\mathsf{input}, \mathsf{ring}]$.
– if $\mathtt{evaluations}[\mathsf{input}, W]$ is not defined, set $\mathtt{evaluations}[\mathsf{input}, W] \leftarrow\$ \mathcal{S}_{eval}$, $\mathtt{anonymous\_key\_map}[\mathsf{input}, W] = \mathsf{pk}_{\mathsf{Sim}}$. Set $\mathtt{evaluations}[\mathsf{input}, W] = y$.
– otherwise, set $y = \mathtt{evaluations}[\mathsf{input}, W]$.

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

Fig. 1. Functionality $\mathcal{F}_{\mathsf{rvrf}}$.

the discrete logarithm h of $H_{\mathbf{G}}(m)$ from the $H_{\mathbf{G}}$'s database and obtains $X^* = h^{-1}W$. If $\mathsf{secret\_keys}[X^*]$ is not empty, it replies by a randomly selected value from $\mathbb{F}_p$. Otherwise, Sim checks if $\mathsf{public\_keys}[X^*]$ exists to see whether a corresponding public key of X^* exists. If it does not exist, Sim samples randomly a key pk^* which is not stored in public_keys and stores $\mathsf{public\_keys}[X^*] = \mathsf{pk}^*$. In any case, it obtains $\mathtt{evaluations}[m, W]$ by sending a message $(\mathsf{eval}, \mathsf{sid}, \mathsf{pk}^*, W, m)$ and replies with $\mathtt{evaluations}[m, W]$. 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 $\mathtt{evaluations}[m, W]$ 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 $\mathtt{anonymous\_key\_map}[m, W'] = \mathsf{pk}^*$. 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, Sim never aborts during the simulation of H.
We note that the anonymous keys for honest parties generated by $\mathcal{F}_{\mathsf{rvrf}}$ are independent from their keys. Therefore, if $X^* = h^{-1}W$ is an honest verification key, Sim returns a random value because $\mathtt{evaluations}[m, W]$ is not defined or will not be defined in $\mathcal{F}_{\mathsf{rvrf}}$ in this case except with a negligible probability. If it ever happens i.e., if $\mathcal{F}_{\mathsf{rvrf}}$ selects randomly $W = hX^*$, $\mathcal{Z}$ distinguishes the simulation via honest signature verification in the real world. So, this case is covered in our simulation in Figure 4.

```
Oracle $H_{\mathbf{G}}$
Input: $m$
if
$\mathtt{oracle\_queries\_gg}[m] = \perp$
  $h \leftarrow\$ \mathbb{F}_p$
  $P \leftarrow hG$
  $\mathtt{oracle\_queries\_gg}[m] := h$
else:
  $h \leftarrow \mathtt{oracle\_queries\_gg}[m]$
  $P \leftarrow hG$
return inbase
```

Fig. 2. The random oracle $H_{\mathbf{G}}$

```
Oracle $H$
Input: $m, W$
if $\mathtt{oracle\_queries\_h}[m, W] \neq \perp$
  return $\mathtt{oracle\_queries\_h}[m, W]$
$P \leftarrow H_{\mathbf{G}}(m)$
$h \leftarrow \mathtt{oracle\_queries\_gg}[m]$
$X^* := h^{-1}W$ // candidate commitment key
if $\mathsf{secret\_keys}[X^*] = \perp$
  if $\mathsf{public\_keys}[X^*] = \perp$
    $\mathsf{pk}^* \leftarrow\$ \mathbf{G}$
    $\mathsf{public\_keys}[X^*] \leftarrow \mathsf{pk}^*$
  send $(\mathsf{eval}, \mathsf{sid}, W, \mathsf{public\_keys}[X^*], m)$ to $\mathcal{F}_{\mathsf{rvrf}}$
  if $\mathcal{F}_{\mathsf{rvrf}}$ ignores: ABORT
  receive $(\mathsf{evaluated}, \mathsf{sid}, W, m, y)$ from $\mathcal{F}_{\mathsf{rvrf}}$
  $\mathtt{oracle\_queries\_h}[m, W] := y$
else:
  $y \leftarrow\$ \mathbb{F}_p$
  $\mathtt{oracle\_queries\_h}[m, W] := y$
return $\mathtt{oracle\_queries\_h}[m, W]$
```

Fig. 3. The random oracle H

The simulation of the random oracle H_p (See Figure 4) checks whether the random oracle query $(\mathsf{ad}', \mathsf{input}, \mathsf{compk}, W, R, R_m)$ is an $\mathcal{R}_{eval}$ verification query before answering the oracle call. For this, it checks whether $\mathcal{F}_{\mathsf{rvrf}}$ has a recorded valid signature for the message m and the ring ring with the anonymous key W. If there exists such valid signature where compk is part of it, Sim checks whether the first proof of the signature (c, s_1, s_2) generates R, R_m as in rVRF.Verify in order to make sure that it is a $\mathcal{R}_{eval}$ verification query. If it is the case, it assigns c as an answer of $H_p(\mathsf{ring}, m, W, \mathsf{compk}, R, R_m)$ so that $\mathcal{R}_{eval}$ verifies. However, if this input has already been set to another value which is not equal to c or W is a pre-output of an honest key, then Sim aborts because the output of the real world for this signature and the ideal world will be different.
We remind that if an anonymous key W of an honest party for a message m sampled by $\mathcal{F}_{\mathsf{rvrf}}$ equals to a pre-output generated by rVRF.Sign for the same honest party's key and the message m, then $\mathcal{Z}$ can distinguish the ideal and real world outputs because the evaluation value in the ideal world

and real world for m, W will be different because of the simulation of the random oracle H i.e., $\texttt{oracle\_queries\_h}[m, W] \neq \texttt{evaluations}[m, W]$. Therefore, Sim aborts if it is ever happen.

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

**parse** $\mathsf{ad}'$ as $\mathsf{ad} \mathbin{+\!\!+} \pi_{\mathtt{ring}} \mathbin{+\!\!+} \mathsf{comring}$
**send** $(\mathsf{request\_signatures}, \mathsf{sid}, \mathsf{ad}, W, \mathsf{input})$
**receive** $(\mathsf{signatures}, \mathsf{sid}, \mathsf{input}, \mathcal{L}_\sigma)$
**if** $\exists \sigma \in \mathcal{L}_\sigma$ where $\mathsf{compk} \in \sigma$ **and** $\mathsf{NIZK}_{\mathcal{R}_{\mathtt{ring}}}.\mathsf{Verify}((\mathsf{compk}, \mathsf{comring}); \pi_{\mathtt{ring}}) \to 1$
  **get** $\pi_1 = (c, s_1, s_2) \in \sigma$
  **if** $R = s_1 G + s_2 K - c\mathsf{compk}, R_m = s_1 H_{\mathbf{G}}(m) - cW$
    $h := \texttt{oracle\_queries\_gg}[m, W]$
    **if** $\texttt{oracle\_queries\_h\_}\mathsf{CP}[\mathsf{ad}, m, \mathsf{compk}, W, R, R_m] = \perp$
        $\texttt{oracle\_queries\_h\_}\mathsf{CP}[\mathsf{ad}, m, \mathsf{compk}, W, R, R_m] := c$
    **else if** $(\texttt{oracle\_queries\_h\_}\mathsf{CP}[\mathsf{ad}, m, \mathsf{compk}, W, R, R_m] \neq c$
    **or** $X^* = h^{-1}W \in \texttt{honest\_keys})$: ABORT
**if** $\texttt{oracle\_queries\_h\_}\mathsf{CP}[\mathsf{ad}, m, \mathsf{compk}, W, R, R_m] = \perp$
  $c \leftarrow\!\!\$\ \mathbb{F}_p$
  $\texttt{oracle\_queries\_h\_}\mathsf{CP}[\mathsf{ad}, m, \mathsf{compk}, W, R, R_m] := c$
**return** $\texttt{oracle\_queries\_h\_}\mathsf{CP}[\mathsf{ad}, m, \mathsf{compk}, W, R, R_m]$
```

Fig. 4. The random oracle H_p

– [**Simulation of** verify] Upon receiving $(\mathsf{verify}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, \mathsf{input}, \sigma)$ from the functionality $\mathcal{F}_{\mathsf{rvrf}}$, Sim runs 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 $h = \texttt{oracle\_queries\_gg}[m]$. 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 public_keys and sets $\mathsf{public\_keys}[X] = \mathsf{pk}$. Then sends $(\mathsf{verified}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, m, \sigma, b_{\mathsf{Sim}}, \mathsf{public\_keys}[X])$ to $\mathcal{F}_{\mathsf{rvrf}}$ and receives back $(\mathsf{verified}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, m, \sigma, y, b)$.
    * 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.
      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 $\mathsf{counter}[m, \mathsf{ring}] > |\mathsf{ring}_m|$. If $\mathcal{F}_{\mathsf{rvrf}}$ is in C3-1, it means that pk belongs to an honest party but this honest party never signs m 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 $\texttt{anonymous\_key\_map}[m, W'] = \mathsf{pk}$. If $[m, W']$ is stored before, it means that Sim obtained $W' = hX$ where $h = \texttt{oracle\_queries\_h}[m]$ but it is impossible to happen since $W = hX$.
    * If $b = b_{\mathsf{Sim}}$, it sets $\texttt{oracle\_queries\_h}[m, W] = y$, 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 $(\mathsf{verified}, \mathsf{sid}, \mathsf{ring}, W, \mathsf{ad}, m, \sigma, \perp, 0)$.

Now, we need to show that the outputs of honest parties in the ideal world are indistinguishable from the honest parties in the real world.

Lemma 5. *Assuming that the DDH problem is hard on the group structure* $(\mathbf{G}, G, K)$, *the outputs of honest parties in the real protocol rVRF are indistinguishable from the output of the honest parties in* $\mathcal{F}_{\mathsf{rvrf}}$.

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 $\mathbf{G}$ and pre-output W generated by $\mathsf{rVRF.Sign}$ are indistinguishable given pk.

Case 1 ($\mathsf{pk} \neq xG$)**:** If $\mathsf{pk} \neq xG$, then pk is uniformly random and independent from x because of the assumption of perfectly hiding. Therefore, the ideal world honest signatures from the real world honest signatures are indistinguishable.

Case 2 ($\mathsf{pk} = xG$)**:** 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 of the ideal protocol 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 $X, Y, Z \in \mathbf{G}$ from the DDH game and simulates the game against $\mathcal{D}$ as follows. 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. $\mathcal{B}$ publishes $\mathbf{G}, G = Y, K$ 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 a message m, it obtains $\mathsf{inbase} = H_{\mathbf{G}}(m) = hY$ from $\mathtt{oracle\_queries\_gg}$ and lets $W = hZ$. Then it lets $\mathsf{compk} = X + \mathsf{b}K$, lets $\pi_{\mathsf{eval}} \to \mathsf{NIZK}_{\mathcal{R}_{eval}}.\mathsf{Simulate}(\mathsf{compk}, W, H_{\mathbf{G}}(m))$ and $\pi_{ring} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{\mathtt{ring}}}.\mathsf{Simulate}((\mathsf{comring}, \mathsf{compk}))$. Remark that if (X, Y, Z) is a DH triple (i.e., $\mathsf{DH}(X, Y, Z) \to 1$), P_{i+1} is simulated as in rVRF because $W = x\mathsf{inbase}$ in this case. Otherwise, P_{i+1} is simulated as in the ideal 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.

Lemma 6. *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 $\mathbf{G}$, $H_{\mathbf{G}}, H, H_p, H_{\mathsf{ring}}$ are random oracles,* $\mathsf{NIZK}_{\mathcal{R}_{eval}}, \mathsf{NIZK}_{\mathcal{R}_{\mathtt{ring}}}$ *are knowledge sound and T_{key} is computationally indistinguishable and binding.*

Proof. The simulation against the real world adversary $\mathcal{A}$ is identical to the real protocol except the output of the honest parties and cases where Sim aborts. We show that the abort cases happen with a negligible probability during the simulation. Sim aborts during the simulation of random oracles H and H_p and during the simulation of verification. We have already explained that the abort case during the simulation of H cannot happen. The abort case happens in the simulation of H_p if $W = hX$ where $X = xG$ or if $\mathtt{oracle\_queries\_h\_CP}[\mathsf{ad}', \mathsf{input}, \mathsf{compk}, W, R, R_m]$ has already been defined by a value which is different than c. The first case happens in H_p if $\mathcal{F}_{\mathsf{rvrf}}$ selects a random $W \in \mathbf{G}$ for an anonymous key of m, pk for the honest party with the other key X and the random oracle $H_{\mathbf{G}}$ selects a random $h \in \mathbb{F}_p$ where $H_{\mathbf{G}}(m) = hG$ and $W = hX$. Clearly, this can happen with a negligible probability in λ. The second case happens in H_p if $\mathcal{A}$ queries with the input $(\mathsf{ad}', \mathsf{input}, \mathsf{compk}, W, R, R_m)$ before $(\pi_1, \pi_2, \mathsf{compk}, \mathsf{comring}, W)$ generated by Gen_{sign}

where $\mathsf{comring}$ is defined in ad'. Since compk is a perfectly hiding Pedersen commitment, thus random and independent from keys, the probability that $\mathcal{A}$ guesses compk before it is generated is negligible. Now, we are left with the abort case 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 $\mathbf{G}$ with the challenges $G, U, V \in \mathbf{G}$. 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 $(\mathbf{G}, p, G, K)$ as a public parameter of the ring VRF protocol to $\mathcal{A}$.

Whenever $\mathcal{B}$ needs to generate a ring signature for m 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 $\mathsf{Gen}_{sign}(\mathsf{ring}, W, \mathsf{pk}, \mathsf{ad}, m)$

```
1: $\mathsf{compk} \leftarrow\!\!\$\ \mathbf{G}$
2: $\pi_{\mathtt{eval}} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{eval}}.\mathsf{Simulate}(\mathsf{compk}, W, m)$
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 $\sigma = (\pi_{eval}, \pi_{ring}, \mathsf{compk}, \mathsf{comring}, W)$
```

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 $\mathtt{evaluations}[m, W]$ it queries m, 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 $H(m, W)$ to $\mathcal{F}_{\mathsf{rvrf}}$ but it does not change the simulation 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 $H_{\mathbf{G}}$ by $\mathcal{B}$ returns hU instead of hG.

During the simulation, when $\mathcal{A}$ outputs a signature $\sigma = (\mathsf{compk}, \pi_{ring}, W, \pi_{eval}, \mathsf{comring})$ of message m with ad which is not recorded in $\mathcal{F}_{\mathsf{rvrf}}$'s record, $\mathcal{B}$ runs $\mathsf{rVRF.Verify}(\mathsf{comring}, m, \mathsf{ad}, \sigma)$. If it verifies, it finds the corresponding ring ring of $\mathsf{comring}$ by checking the random oracle H_{ring}'s database. Remark that there exists ring where Merkle tree root of ring is $\mathsf{comring}$ because if it was not the case σ would not verify which also checks π_{ring}. Then it runs the extractor algorithm of $\mathsf{NIZK}_{\mathcal{R}_{\mathtt{ring}}}$ and obtains $b, \mathsf{opring}, \mathsf{sk}, r$. Then, it computes $X = \mathsf{compk} - \mathsf{b}\,K$. If $\mathsf{pk} = \mathsf{rVRF.OpenRing}(\mathsf{comring}, \mathsf{opring})$ 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 $(\hat{\mathsf{sk}}, \hat{\mathsf{b}})$ such that $\mathsf{compk} = \hat{\mathsf{sk}}G + \hat{\mathsf{b}}\,K$ and $W = \hat{\mathsf{sk}}H_{\mathbf{G}}(m)$. If $W \notin \mathcal{W}[m, \mathsf{ring}]$, $\mathcal{B}$ increments $\mathsf{counter}[m, \mathsf{ring}]$ and adds W to $\mathcal{W}[m, \mathsf{ring}]$.

If X is generated by $\mathcal{B}$ during a key generation process of an honest party and $X = \hat{\mathsf{sk}}G$, $\mathcal{B}$ solves the CDH problem as follows: $W = \hat{\mathsf{sk}}hU$ where $h = \mathtt{oracle\_queries\_gg}[m]$. Since $X = rV$, $W = \hat{\mathsf{sk}}huG = rhuV$. 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 $\sigma_j = (\mathsf{compk}_j, \pi_{ring_j}, W_j, \pi_{eval_j}, \mathsf{comring}) \in \{\sigma_i\}_{i=1}^{t'}$, $\mathcal{B}$ runs the extractor for $\mathcal{R}_{\texttt{ring}}$ and obtains $\mathsf{opring}_j, \mathsf{b}_j, \mathsf{sk}_j, r_j$. Then it obtains the public key $\mathsf{pk}_j = \mathsf{rVRF.OpenRing}(\mathsf{ring}, \mathsf{opring}_j)$ where $\mathsf{pk}_j \in \mathsf{ring}$ and $X_j = \mathsf{compk} - \mathsf{b}K$ [6]. Then it adds X_j to a list $\mathcal{X}$ and pk_j to a set $\mathcal{PK}$. One of the following cases happens:

- All X_j in $\mathcal{X}$ are different and $|\mathcal{PK}| \leq t$: Each $\mathsf{pk}_j \in \mathcal{PK}$ commits to a secret key sk_j such that $\mathsf{sk}_j = \mathsf{Com.Open}(\mathsf{pk}_j, r_j)$. If all X_j's are different, then there exists a $\mathsf{pk}_j \in \mathcal{PK}$ where $\mathsf{sk}_j = \mathsf{Com.Open}(\mathsf{pk}_j, r_j)$ and $\mathsf{sk}'_j = \mathsf{Com.Open}(\mathsf{pk}_j, r'_j)$ such that $\mathsf{sk}_j G, \mathsf{sk}'_j G \in \mathcal{X}$. 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. We note that this case does not happen if $\mathsf{rVRF.KeyGen}$ outputs $\mathsf{pk} = \mathsf{sk}G$.
- 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. This cannot happen because $\mathcal{B}$ solves the CDH when $\mathcal{A}$ outputted σ_a.
- There exists 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 $(\hat{\mathsf{sk}}_a, \hat{\mathsf{b}}_a)$ and $(\hat{\mathsf{sk}}_b, \hat{\mathsf{b}}_b)$, respectively such that $\mathsf{compk}_a = \hat{\mathsf{sk}}_a G + \hat{\mathsf{b}}_a K \mathsf{compk}_b = \hat{\mathsf{sk}}_b G + \hat{\mathsf{b}}_b K$ and $W_a = \hat{\mathsf{sk}}_a H_{\mathbf{G}}(m), W_b = \hat{\mathsf{sk}}_b H_{\mathbf{G}}(m)$. Since $W_a \neq W_b$, $\hat{\mathsf{sk}}_a \neq \hat{\mathsf{sk}}_b$. So, $\mathcal{B}$ can obtain two different and non trivial representation of $X_a = X_b$ i.e., $X_a = X_b = \hat{\mathsf{sk}}_a G + (\hat{\mathsf{b}}_a - \mathsf{b}_a) K = \hat{\mathsf{sk}}_b G + (\hat{\mathsf{b}}_b - \mathsf{b}_b) K$. Thus, $\mathcal{B}$ finds the discrete logarithm of $K = U$ in base G which is $u = \frac{\hat{\mathsf{sk}}_a - \hat{\mathsf{sk}}_b}{\hat{\mathsf{b}}_a - \mathsf{b}_a - \hat{\mathsf{b}}_b + \mathsf{b}_b}$. $\mathcal{B}$ 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 SpecialG

Theorem 5. *Our specialized* rVRF *over* pp_{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 7 and* $\mathsf{NIZK}_{\mathcal{R}_{eval}}$ *is zero-knowledge and knowledge sound, the decisional Diffie-Hellman (DDH) problem are hard in* $\mathbf{G}$ *(so the CDH problem is hard as well) and the commitment scheme* Com *is binding and perfectly hiding.*

Before we start to analyse our protocol, we should define the algorithm Gen_{sign} for $\mathcal{F}_{\mathsf{rvrf}}$ and show that $\mathcal{F}_{\mathsf{rvrf}}$ with Gen_{sign} satisfies the anonymity defined in Definition 6. It is very similar to Algorithm 1.

Lemma 7. $\mathcal{F}_{\mathsf{rvrf}}$ *running Algorithm 2 satisfies anonymity defined in Definition 6 assuming that* ZKCont *is a zero-knowledge as defined in Definition 7.*

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 $\mathsf{ZKCont.Simulate}(\mathsf{td}, \mathsf{comring}, \mathcal{R}^{\texttt{inner}}_{\texttt{ring}})$. 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.

Theorem 6. rVRF *over* pp_{rvrf} *realizes* $\mathcal{F}_{\mathsf{rvrf}}$ *running* Gen_{sign} *in Algorithm 1 [11,12] in the random oracle model assuming that* $\mathsf{NIZK}_{\mathcal{R}_{eval}}$ *and* $\mathsf{NIZK}_{\mathcal{R}_{ring}}$ *are zero-knowledge and knowledge sound, the decisional Diffie-Hellman (DDH) problem are hard in* $\mathbf{G}$ *(so the CDH problem is hard as well) and the commitment scheme* Com *is binding and perfectly hiding.*

[6] The reason that we do not use sk_j while computing X_j is that to show that $\mathcal{R}_{\texttt{ring}}$ does not need (sk_j, r_j) as a witness if $\mathsf{rVRF.KeyGen}$ generates pk as $\mathsf{sk}G$. So, we want to show that $\mathcal{B}$ can solve CDH even if sk is not the part of the witness of $\mathcal{R}_{\texttt{ring}}$

The security proof follows the same proof we have for Theorem 6 except the parts that we run extractor and simulator algorithms in $\mathsf{NIZK}_{\mathcal{R}_{\mathtt{ring}}}$ and $\mathsf{NIZK}_{\mathcal{R}_{eval}}$. We construct the same Sim described in the proof of Theorem 6 because it does not deploy any extractor or simulator of NIZK for $\mathcal{R}_{eval}$ and $\mathcal{R}_{\mathtt{ring}}$. Similarly, Lemma 5 applies here. The only difference is in Lemma 6 since Gen_{sign} is different than Algorithm 1. We first replace Gen_{sign} run by $\mathcal{B}$ in Algorithm 3 defined for Lemma 6 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 $\sigma = (\pi_{eval}, \pi_{ring}, \mathsf{compk}, \mathsf{comring}, W)$
```

The other change is that we replace all extractors in Lemma 6 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 6 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 $\mathsf{sk}G$. When $\mathcal{B}$ sees a signature $\sigma = (\mathsf{compk}, \pi_{ring}, W, \pi_{eval}, \mathsf{comring})$ of input, it runs the extractor for $\mathsf{NIZK}_{\mathcal{R}_{\mathtt{rvrf}}}$ and obtains $\mathsf{sk}, r, \mathsf{opring}$. Then, it lets X be $\mathsf{sk}G$. If X is generated for an honest party, it solves the CDH as described in Lemma 6 for the same case. If $\mathcal{W}[\mathsf{input}, \mathsf{ring}] = t' > |\mathsf{ring}_{mal}| = t$, it runs the extractors for $\mathsf{NIZK}_{\mathcal{R}_{\mathtt{ring}}}$ of all malicious signatures of input for ring and obtains $\{(\mathsf{sk}_j, r_j, \mathsf{opring}_j)\}_{j=1}^{t'}$. Then, for all $j \in [1, t']$, it adds $X_j = \mathsf{sk}_j G$ 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 6 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 6 does not happen.

B 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 1.

B.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 $\mathcal{F}_{zk}$ [25] 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.

$\mathcal{F}_{\mathsf{rvrf}}^{s}$ 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 $\mathsf{Gen}_{\eta}(m, \mathsf{pk}_i, y) \rightarrow \eta$ 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 $\mathsf{Gen}_{\eta}(m, \mathsf{pk}_i, y) \rightarrow \eta$ 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 $\mathsf{Gen}_{\pi}(W, m) \rightarrow \pi$ 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 $\mathcal{F}_{\mathsf{rvrf}}^{s}$.