Ethical identity, ring VRFs, and zero-knowledge continuations

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

Web 3.0 Foundation

December 28, 2022

Abstract. We introduce a new cryptographic primitive, aptly named *ring verifiable random functions (ring VRF)*, Ring VRFs are (anonymized) ring signatures that prove correct evaluation of an authorized signer's PRF, while hiding the specific signer's identity within some set of possible signers, known as the ring.

We discover a family of ring VRF protocols with surprisingly efficient instantiations, thanks to our novel *zero-knowledge continuation* technique. Intuitively our ring VRF signers generate two linked proofs, one for PRF evaluation and one for ring membership. An evaluation proof needs only a cheap Chaum-Pedersen DLEQ proof, while ring membership proof depends only upon the ring itself. We reuse this ring membership proof across multiple inputs by expanding a Groth16 trusted setup to rehide public inputs when rerandomizing the Groth16. Incredibly, 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.

We discuss applications that range across the anonymous credential space:

As in Bryan Ford's proof-of-personhood work, a ring VRF output acts like a unique pseudonymous identity within some desired context, given as the ring VRF input, but remains unlinkable between different contexts. These unlinkable but unique pseudonyms provide a better balance between user privacy and service provider or social interests than attribute based credentials like IRMA credentials.

Ring VRFs support anonymously rationing or rate limiting resource consumption that winds up vastly more flexible and efficient than purchases via money-like protocols.

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

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.

Ring VRFs: A ring signature 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 pseudorandom function (PRF) defined by the actual signer's key. Ring VRF outputs then provide linking proofs between different signatures iff 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 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) [9,10] model. We show that our ring VRF protocol is secure in the UC model.

In §5, 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 [19] admit a transformation of a valid proof into another valid but unlinkable proof of the exact same statement. In practice, rerandomization never gets deployed because the public inputs link different usages, breaking privacy.

We demonstrate in §5 a simple transformation of any Groth16 zkSNARK into a *zero-knowledge continuation* whose public inputs involve opaque Pedersen commitments, with cheaply rerandomizable blinding factors and 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 [8].

As recursive SNARKs might remain slow, we expect zero-knowledge continuations via rerandomization become essential for zkSNARKs used in identity and elsewhere outside the cryptocurrency 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 §8).

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 [7] 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 is not merely the most common use cases for anonymous credentials, but in fact 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 [27], but broadly speaking they all bring matching harmful uses.

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 Bryan Ford's work on proof-of-personhood parties [16,5] 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 [7] or group signatures [24]. 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 [12], or else from OPRFs like PrivacyPass [14], 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 §9)

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 [14] 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.

In this, we need authenticated domain separation of products or identity consumers in queries to users' ring VRF credentials. We briefly discuss some sensible patterns in §9.2 below, but overall authenticated domain separation resemble TLS certificates except simpler in that roots of trust can self authenticate if root keys act as domain separators.

Acknowledgments: We thank Oana Ciobotaru for her repeated close readings of the paper.

1 Protocol 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, and give some intuition for our later amortization trick.

As VRFs do [26], all ring VRFs need $\mathsf{rVRF.KeyGen}$ algorithm, which creates a random secret key sk and associated public key pk, as well as a deterministic evaluation algorithm

- $\mathsf{rVRF.Eval} : (\mathsf{sk}, \mathsf{input}) \mapsto \mathsf{out}$ which computes the VRF output out from a secret key sk and a message input.

Our KeyGen and Eval initially resemble EC VRFs like [28,29,17]. We demand pseudo-randomness properties from Eval, which could mirror [26] if desired. We provide a UC definition resembling [13,2] which handles adversarial keys better however.

Ring VRFs differ from VRFs in that they do not expose a specific signer, and instead prove the signer's key lies in some plausible signer set ring, much like how ring signatures differ from signatures. Ring VRFs differ from ring signatures in that they prove a VRF output out.

At their simplest, ring VRFs' other algorithms might operate directly upon the plausible signer set ring, like:

- $\mathsf{rVRF.Sign} : (\mathsf{sk}, \mathsf{ring}, \mathsf{input}) \mapsto \sigma$ 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 $\perp$.

After success, our verifier should be convinced that $\mathsf{pk} \in \mathsf{ring}$, that $\mathsf{out} = \mathsf{rVRF.Eval}(\mathsf{sk}, \mathsf{input})$ for some $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}$, and that out is pseudo-random. In other words, this simplified ring VRF could be instantiated by making Eval a pseudo-random (hash) function, and using a NIZK for a language like

$$\{\, \mathsf{out}, \mathsf{input}, \mathsf{ring} \mid \exists (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}, \quad \mathsf{pk} \in \mathsf{ring}, \quad \mathsf{out} = \mathsf{Eval}(\mathsf{sk}, \mathsf{input}) \,\}$$

Zero-knowledge here says our verifier learns nothing about the specific signer, except that their key is in the ring and maps $\mathsf{input} \mapsto \mathsf{out}$. Importantly, pseudo-randomness also says that out is an identity for the specific signer, but only within the context of input.

Aside from proving a PRF evaluation, we always need Sign and Verify to sign some associated data ass, as otherwise the ring VRF signature become unmoored and permits replay attacks. As an example, our identity protocol below in §8 yields the same ring VRF outputs each time the same user logs into the same site, which suffers replay attacks unless ass 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 $(\mathsf{input}, \mathsf{ass})$. As an example, Praos [13] binds their VRF public key together with a second public key for another (forward secure) signature scheme, with which they sign their ass, the block itself. An EC VRF should expose an ass

parameter which it hashes when computing its challenge hashes. Aside from saving redundant signatures, exposing ass avoids user key handling mistakes that create replay attacks.

Ring VRFs cannot so easily be combined with another signatures, which makes ass essential,[1] but thankfully our ring VRFs expose ass exactly like EC VRFs do in §4.[2]

We would need time $O(|\mathsf{ring}|)$ in Sign and Verify merely to read this ring argument though, which severely limits applications. Instead, ring signatures run asymptotically faster by replacing the ring argument with some set commitment to ring, roughly like what ZCash does [22].

- $\mathsf{rVRF.CommitRing} : (\mathsf{ring}, \mathsf{pk}) \mapsto (\mathsf{comring}, \mathsf{opring})$ returns a commitment for a set ring of public keys, and optionally the operning opring for some $\mathsf{pk} \in \mathsf{ring}$ 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 language and NIZK by the by the opening condition

$$\exists \mathsf{opring} \text{ s.t. } \mathsf{pk} = \mathsf{OpenRing}(\mathsf{comring}, \mathsf{opring}).$$

After these corrections, our algorithms for a *ring verifiable random function with associated data* (rVRF-AD) now really look like

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

Although an asymptotic improvement, our opening $\mathsf{OpenRing}$ invariably still winds up being extremely heavy inside a zkSNARK. We solve this obstacle in §5 below by introducing *zero-knowledge continuations*, a new zkSNARK technique built from rerandomizable Groth16s [19] and designed for composition and reuse.

In this, we should split the language describing our ring VRF into a VRF evaluation language $L_{\mathtt{eval}}$ and a reusable or continuable language $L_{\mathtt{ring}}$, which enforces our heavy $\mathsf{pk} = \mathsf{OpenRing}(\mathsf{comring}, \mathsf{opring})$ condition. Anonymity requires we rerandomize a Groth16 SNARK for $L_{\mathtt{ring}}$ ala [3, Theorem 3, Appendix C, pp. 31]. Yet, we must join aka pass pk from $L_{\mathtt{ring}}$ to $L_{\mathtt{eval}}$ somehow, which demands some hiding commitment compk of course.

$$L_{\mathtt{eval}} = \{ \mathsf{out}, \mathsf{input}, \mathsf{ass}, \mathsf{compk} \mid \mathsf{out} = \mathsf{PRF}(\mathsf{sk}, \mathsf{input}), \mathsf{compk} \text{ commits to } \mathsf{sk} \}$$

$$L_{\mathtt{ring}} = \{ \mathsf{compk}, \mathsf{comring} \mid \mathsf{compk} \text{ commits to } \mathsf{pk} = \mathsf{OpenRing}(\mathsf{comring}, \mathsf{opring}) \}$$

We discovered this becomes incredibly efficient if one specializes the Groth16 construction: An inner true Groth16 for $L^{\mathtt{inner}}_{\mathtt{ring}}$ handles the secret key sk directly, while our Pedersen commitment compk is built outside this Groth16.

$$L^{\mathtt{inner}}_{\mathtt{ring}} = \{ \mathsf{sk}, \mathsf{comring} \mid (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}, \mathsf{pk} = \mathsf{OpenRing}(\mathsf{comring}, \mathsf{opring}) \}$$

Our zero-knowledge continuation in §5 rerandomizes $\mathsf{compk} = \mathsf{pk} + b\,K$ without reproving the Groth16 $L^{\mathtt{inner}}_{\mathtt{ring}}$. For this, the secret key sk must be a public input of $L^{\mathtt{inner}}_{\mathtt{ring}}$, and the Groth16 trusted setup must be expanded by a multiple of the otherwise independent point K. In §4, we introduce an extremely efficient NIZK for $L_{\mathtt{eval}}$, which also provides an essential proof-of-knowledge for compk.

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

[2] We suppress multiple input-output pairs until §9.3 below, but they work like in [14] too.

2 Background

We briefly establish elliptic curve notion and recall some standard definitions and assumptions.

2.1 Elliptic curves

We obey mathematical and cryptographic implementation convention by adopting additive notation for elliptic curve and multiplicative notation for elliptic curve scalar multiplications. All objects implicitly depend a security parameter λ.

We have an elliptic curve $\mathbb{G}$ over a field of characteristic q, equipped with a type III pairing $e : \mathbf{G}_1 \times \mathbf{G}_2 \to \mathbf{G}_T$, where the groups $\mathbf{G}_1 \leq \mathbb{G}[\mathbb{F}_q]$, $\mathbf{G}_2 \leq \mathbb{G}[\mathbb{F}_{q^2}]$, and $\mathbf{G}_T \leq \mathbb{F}^*_{q^{12}}$ all have prime order $p \approx 2^{2\lambda}$.

We write $\mathbf{G}$ when discussing the Chaum-Pedersen DLEQ proofs, which do not employ pairings, but $\mathbf{G}$ always denotes $\mathbf{G}_1$ eventually. We avoid pairing unfriendly assumptions like DDH of course, but really we employ the algebraic group model (AGM) throughout.

We sweep cofactor concerns under the rug when discussing Groth16, where pairings demand deserialization prove group membership in $\mathbf{G}_1$ or $\mathbf{G}_2$. We explicitly multiply by the cofactor h when doing Chaum-Pedersen DLEQ proofs though, as not doing so risks miss-readings by implementers.

We also let $\mathbb{J}$ denote a ZCash Sapling style "JubJub" Edwards curve over $\mathbb{F}_p$, with distinguished subgroup $\mathbf{J}$ of prime order $p_{\mathbf{J}}$, so that SNARKs on $\mathbb{G}$ prove $\mathbf{J}$ arithmetic relatively cheaply. Aside from Jubjub, we optionally want a "sister" Edwards curve $\mathbb{G}$, with a subgroup $\mathbf{G}$ of the same order p as $\mathbf{G}_1$, but which lacks any pairing.

All our security proofs ignore these underlying elliptic curve details: AGM is used for $\mathbb{G}$ in Groth16 sections. $\mathbf{G}_1$, $\mathbf{G}'$, and $\mathbf{J}$ always have a hard DDH problem and their cofactor are ignored.

We let $H_p : \{0,1\}^* \to \mathbb{F}_p$ and $H_{\mathbf{G}'} : \{0,1\}^* \to \mathbf{G}_1$ denote a hash-to-scalar and a hash-to-curve with ranges $\mathbb{F}_p$ and $\mathbf{G}'$, respectively, always modeled as random oracles.

2.2 Zero-knowledge proofs

We let $\mathcal{R}$ denote a polynomial time decidable relation, so the language $\mathcal{L} = \{x \mid \exists \omega (x; \omega) \in \mathcal{R}\}$ lies in NP. All non-interactive zero-knowledge proof systems have some setup procedure `Setup` that takes some implicit parameters and some "circuit" description of $\mathcal{R}$, and may produces a structured reference string (SRS). We discuss SRSes and their toxic waste in §5 but SRSes remain implicit in our notation.

A non-interactive proof system for $\mathcal{L}$ consists of Prove and Verify PPT algorithms

- $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Prove}(x;\omega) \mapsto \pi$ creates a proof π for a witness and statement pair $(x;\omega) \in \mathcal{R}$.
- $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Verify}(x;\pi)$ returns either true of false, depending upon whether π proves x.

which satisfy the following completeness, zero-knowledge, and knowledge soundness definitions.

We always describe circuits as languages $\mathcal{L}$ and write $\mathsf{NIZK}_{\mathcal{L}}$ for two reasons: All SNARK circuits have many logic wires in $\mathcal{R}$ other than the public input wires x and the secret input witness wires ω. An existential quantifiers $\exists$ more clearly distinguishs public inputs x from secret input witnesses ω than tuple position. We also benefited from language in the preceding informal exposition, which did not always require specifying ω.

Definition 1. *We say* $\mathsf{NIZK}_{\mathcal{R}}$ *is* complete *if* $\mathsf{Verify}(x, \mathsf{Prove}(x;\omega))$ *succeeds for all* $(x;\omega) \in \mathcal{R}$.

Definition 2. *We say* $\mathsf{NIZK}_{\mathcal{R}}$ *is* zero-knowledge *if there exists a PPT simulator* $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Simulate}(x) \mapsto \pi$ *that outputs proofs for statement* $x \in L$ *alone, which are computationally indistinguishable from legitimate proofs by* Prove*, i.e. any non-uniform PPT adversary* V^* *cannot distinguish pairs* $(x;\pi)$ *generated by* Simulate *or by* Prove *except with odds negligible in* λ *(see [3, Def. 9, §A, pap. 29]).*

Definition 3. *We say* $\mathsf{NIZK}_{\mathcal{R}}$ *is* (white-box) knowledge sound *if for any non-uniform PPT adversary* $\mathcal{A}$ *who outputs a statement* $x \in \mathcal{L}$ *and proof* π *there exists a PPT extractor algorithm* Extract *that white-box observes* P^* *and if* $\mathsf{Verify}(x;\pi)$ *holds then* Extract *returns an* ω *for which* $(x;\omega) \in \mathcal{R}$ *(see [3, Def. 7, §A, pap. 29]).*

Our zero-knowledge continuations in §5 demand rerandomizing existing zkSNARKs, which only Groth16 supports [19]. We therefore introduce some details of Groth16 [19] there, when we tamper with Groth16's SRS and `Setup` to create zero-knowledge continuations.

2.3 Universal Composability (UC) Model

We define the security of ring VRFs in the UC model [9,10]. In a nutshell, Canetti [9,10] defines the UC model as follows:

A protocol ϕ in the UC model is an execution between distributed interactive Turing machines (ITM). Each ITM has a storage to collect the incoming messages from other ITMs, adversary $\mathcal{A}$ or the environment $\mathcal{Z}$. $\mathcal{Z}$ is an entity to represent the external world outside of the protocol execution. The environment $\mathcal{Z}$ initiates ITM instances (ITIs) and the adversary $\mathcal{A}$ with arbitrary inputs and then terminates them to collect the outputs. We identify an ITI with its session identity sid and its ITM's identifier pid. In this paper, when we call an entity as a party in the UC model we mean an ITI with the identifier $(\mathsf{sid}, \mathsf{pid})$.

We define the ideal world where there exists an ideal functionality $\mathcal{F}$ and the real world where a protocol ϕ is run as follows:

Real world: $\mathcal{Z}$ initiates ITMs and $\mathcal{A}$ to run the protocol instance with some input $z \in \{0,1\}^*$ and a security parameter λ. After $\mathcal{Z}$ terminates the protocol instance, we denote the output of the real world by the random variable $\mathsf{EXEC}(\lambda, z)_{\phi,\mathcal{A},\mathcal{Z}} \in \{0,1\}$. Let $\mathsf{EXEC}_{\phi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\mathsf{EXEC}(\lambda, z)_{\phi,\mathcal{A},\mathcal{Z}}\}_{z\in\{0,1\}^*}$.

Ideal world: $\mathcal{Z}$ initiates ITMs and a simulator Sim to contact with the ideal functionality $\mathcal{F}$ with some input $z \in \{0,1\}^*$ and a security parameter λ. $\mathcal{F}$ is trusted meaning that it cannot be corrupted. Sim forwards all messages forwarded by $\mathcal{Z}$ to $\mathcal{F}$. The output of execution with $\mathcal{F}$ is denoted by a random variable $\mathsf{EXEC}(\lambda, z)_{\mathcal{F},\mathsf{Sim},\mathcal{Z}} \in \{0,1\}$. Let $\mathsf{EXEC}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}$ denote the ensemble $\{\mathsf{EXEC}(\lambda, z)_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}\}_{z\in\{0,1\}^*}$.

Definition 4 (UC-Security of ϕ). *Given a real world protocol* ϕ *and an ideal functionality* $\mathcal{F}$ *for the protocol* ϕ*, we call that* ϕ *is UC-secure i.e.,* ϕ *UC-realizes* $\mathcal{F}$*, if for all PPT adversaries* $\mathcal{A}$*, there exists a simulator* Sim *such that for any environment* $\mathcal{Z}$*,* $\mathsf{EXEC}_{\phi,\mathcal{A},\mathcal{Z}}$ *is indistinguishable from* $\mathsf{EXEC}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}$.

3 Security of Ring VRFs

In this section, we model the security of ring VRF in both standard model and UC model. We show that our ring VRF protocol is UC secure but we also want to define the security of ring VRF in the standard model for potential protocols which may not be shown secure in the UC-model.

3.1 Standard model

We briefly give security games for a *ring verifiable random function with associated data* rVRF-AD constructions, which broadly resemble existing VRF or ring signature definitions. In this, we include the full key commitment procedure and associated data ass, as they impact implementers and applications, but we suppress multiple inputs for brevity.

Definition 5. *We say an rVRF-AD satisfies* evaluation correctness *if* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}$ *implies* $\mathsf{Eval}(\mathsf{sk}, \mathsf{input}) = \mathsf{Verify}(\mathsf{pk}, \mathsf{input}, \mathsf{ass}, \mathsf{Sign}(\mathsf{sk}, \mathsf{input}, \mathsf{ass}))$*, succeeds and also* ring commitment correctness *if* $\mathsf{pk} \in \mathsf{ring}$ *implies both*

$$\mathsf{OpenRing}(\mathsf{CommitRing}(\mathsf{ring}, \mathsf{pk})) = \mathsf{pk} \quad \textit{and} \quad \mathsf{CommitRing}(\mathsf{ring}) = \mathsf{CommitRing}(\mathsf{ring}, \mathsf{pk}).\mathsf{comring}.$$

We lack anonymity against full key exposure ala [4, pp. 6 Def. 4] of course, due to the VRF output, but instead demand a weaker anonymity condition similar to [4, pp. 5 Def. 3]:

Definition 6. *We let* $\mathcal{O}\mathsf{Sign}$ *denote a ring signature CMA oracle, meaning*

- $\mathcal{O}\mathsf{Sign}(\mathsf{keygen})$ *creates a fresh key pair* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}$*, logs it, and adds* pk *to a master public key set* ring_0 *it maintains, and then returns* pk.
- $\mathcal{O}\mathsf{Sign}(\mathsf{comring}, \mathsf{opring}, \mathsf{input}, \mathsf{ass})$ *returns the ring VRF signature* $\mathsf{Sign}(\mathsf{sk}, \mathsf{opring}, \mathsf{input}, \mathsf{ass})$*, provided it logged* $(\mathsf{pk}, \mathsf{sk})$ *with* $\mathsf{pk} = \mathsf{OpenKey}(\mathsf{comring}, \mathsf{opring})$ *previously.*

Definition 7. *We say* rVRF *satisfies* ring anonymity *if any PPT adversary* $\mathcal{A}$ *has an advantage only negligible in* λ *to win the game:*

> *Initially* $\mathcal{A}$ *outputs a message* input*, associated data* ass*, two distinct public keys* $\mathsf{pk}_0, \mathsf{pk}_1 \in \mathcal{O}\mathsf{Sign}.\mathsf{ring}_0$ *created by* $\mathcal{O}\mathsf{Sign}$*, and a ring* $\mathsf{ring} \subset \mathcal{O}\mathsf{Sign}.\mathsf{ring}_0$ *containing* $\mathsf{pk}_0, \mathsf{pk}_1$*. Set* $\mathsf{comring} = \mathsf{CommitRing}(\mathsf{ring})$*. Next the challenger chooses* $j = 0$ *or* $j = 1$ *and gives* $\mathcal{A}$ *some signature* $\sigma = \mathsf{Sign}(\mathsf{sk}_j, \mathsf{opring}, \mathsf{input}, \mathsf{ass})$ *with* $\mathsf{OpenKey}(\mathsf{comring}, \mathsf{opring}) = \mathsf{pk}_j$*.* $\mathcal{A}$ *calls* $\mathcal{O}\mathsf{Sign}$ *of Definition 6 throughout, except the adversary* $\mathcal{A}$ *loses if they ever query* $\mathcal{O}\mathsf{Sign}(\mathsf{comring}', \mathsf{opring}', \mathsf{input}, \cdot)$ *on* input *for some* $\mathsf{comring}', \mathsf{opring}'$ *with* $\mathsf{OpenKey}(\mathsf{comring}', \mathsf{opring}') \in \{\mathsf{pk}_0, \mathsf{pk}_1\}$*. Finally* $\mathcal{A}$ *guesses* j *and wins if correct.*

We similarly want a ring uniqueness that limits adversaries who know secret keys, as well as a ring unforgeability resembling [4, pp. 7 Def. 7].

Definition 8. *We say* rVRF *satisfies* ring uniqueness *(resp. ring unforgeability)if any PPT adversary* $\mathcal{A}$ *has an advantage only negligible in* λ *to win the game:*

> $\mathcal{A}$ *calls* $\mathcal{O}\mathsf{Sign}$ *of Definition 6 throughout, but also creates its own keys freely. Finally* $\mathcal{A}$ *outputs a set* ring*, a message* input*, and* k *valid ring VRF signatures for* ring *on* input *(resp. and also associated data* ass*). each with distinct outputs. Set* $k' := |\mathsf{ring} \setminus \mathcal{O}\mathsf{Sign}.\mathsf{ring}_0|$*.* $\mathcal{A}$ *wins if* $k > k'$ *and they invoked* $\mathcal{O}\mathsf{Sign}$ *strictly fewer than* $k - k'$ *times on* input *(resp. and also* ass*), and distinct* i *with* $\mathsf{pk}_i \in \mathsf{ring} \cap \mathcal{O}\mathsf{Sign}.\mathsf{ring}_0$*.*

Any ring VRF becomes a non-anonymized VRF whenever the ring becomes a singleton $\mathsf{ring} = \{\mathsf{pk}\}$ of course. We inherit ring VRF output properties from non-anonymized VRFs because they wind up strongest for singleton rings, i.e. $|\mathsf{ring}| = 1$. These include residual pseudo-randomness [26, Def. VRF (3) §3.2, pp. 4] and residual unpredictability [26, Def. VUF (3) §3.2, pp. 5] (also [21, Def. 4, pp. 8]). We caution firstly that pseudo-randomness in [26] handles adversarially generated keys poorly, and secondly that VUFs simplify theoretical exposition ala [21] but this simplification increases miss-use risks in practice.

$\mathcal{F}_{\mathsf{rvrf}}$ 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 $(\mathsf{verificationkey}, \mathsf{sid}, \mathsf{pk})$ from Sim, verify that pk has not been recorded before for sid; then, store in the table `verification_keys`, under P_i, the value pk. Return $(\mathsf{verificationkey}, \mathsf{sid}, \mathsf{pk})$ to P_i.

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

[Malicious Ring VRF Evaluation.] upon receiving a message $(\mathsf{eval}, \mathsf{sid}, \mathsf{pk}_i, W, m)$ from Sim, if pk_i is recorded under an honest party's identity or if there exists $W' \neq W$ where `anonymous_key_map`$[m, W'] = \mathsf{pk}_i$, ignore the request. Otherwise, record in the table `verification_keys` the value pk_i under Sim if pk_i is not in `verification_keys`.

If `anonymous_key_map`$[m, W]$ is not defined before, set `anonymous_key_map`$[m, W] = \mathsf{pk}_i$ and let $y \leftarrow\!\!\$\ \mathcal{S}_{eval}$ and set `evaluations`$[m, W] = y$.

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

[Honest Ring VRF Signature and Evaluation.] upon receiving a message $(\mathsf{sign}, \mathsf{sid}, \mathsf{ring}, \mathsf{pk}_i, m)$ 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 the table `verification_keys`. If that wasn't the case, just ignore the request. If there exists no W' such that `anonymous_key_map`$[m, W'] = \mathsf{pk}_i$, let $W \leftarrow\!\!\$\ \mathcal{S}_W$ and let $y \leftarrow\!\!\$\ \mathcal{S}_{eval}$. If there exists W where `anonymous_key_map`$[m, W]$ is defined, then abort. Otherwise, set `anonymous_key_map`$[m, W] = \mathsf{pk}_i$ and set `evaluations`$[m, W] = y$.

In any case (except ignoring and aborting), obtain W, y where `anonymous_key_map`$[m, W] = \mathsf{pk}_i$ and `evaluations`$[m, W] = y$ and run $\mathsf{Gen}_{sign}(\mathsf{ring}, W, \mathsf{pk}) \to \sigma$. Verify that $[m, W, \mathsf{ring}, \sigma, 0]$ is not recorded. If it is recorded, abort. Otherwise, record $[m, W, \mathsf{ring}, \sigma, 1]$. Return $(\mathsf{signature}, \mathsf{sid}, \mathsf{ring}, W, m, y, \sigma)$ to P_i.

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

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

C1 If there exits a record $[m, W, \mathsf{ring}, \sigma, b']$, set $b = b'$. (This condition guarantees the completeness and consistency.)

C2 Else if `anonymous_key_map`$[m, W]$ is an honest verification key and there exists a record $[m, W, \mathsf{ring}, \sigma', 1]$ for any σ', then let $b = 1$ and record $[m, W, \mathsf{ring}, \sigma, 1]$. (This condition guarantees that if m is signed by an honest party for the ring ring at some point, then the signature is $\sigma' \neq \sigma$ which is generated by the adversary is valid)

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

1. If $W \notin \mathcal{W}$ and $|\mathcal{W}[m, \mathsf{ring}]| > |\mathsf{ring}_{mal}|$ where ring_{mal} is a set of malicious keys in ring, set $b = 0$. (This condition guarantees uniqueness meaning that the number of verifying outputs that Sim can generate for m, ring is at most the number of malicious keys in ring.).
2. Else if $\mathsf{pk}_{\mathsf{Sim}}$ is an honest verification key, set $b = 0$. (This condition guarantees unforgeability meaning that if an honest party never signs a message m for a ring ring)
3. Else if there exists $W' \neq W$ where `anonymous_key_map`$[m, W'] = \mathsf{pk}_{\mathsf{Sim}}$, set $b = 0$. (This condition guarantees that there exists a unique anonymous key for each $(m, \mathsf{pk}_{\mathsf{Sim}})$)
4. Else set $b = b_{\mathsf{Sim}}$.

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

- if $W \notin \mathcal{W}[m, \mathsf{ring}]$, add W to $\mathcal{W}[m, \mathsf{ring}]$.
- if $\mathsf{pk}_{\mathsf{Sim}}$ is not recorded, record it in `verification_keys` under Sim.
- if `evaluations`$[m, W]$ is not defined, set `evaluations`$[m, W] \leftarrow\!\!\$\ \mathcal{S}_{eval}$, `anonymous_key_map`$[m, W] = \mathsf{pk}_{\mathsf{Sim}}$. Set $y =$ `evaluations`$[m, W]$.
- otherwise, set $y =$ `evaluations`$[m, W]$.

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

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

3.2 UC model

In this section, we define the security of a ring VRF protocol in the UC model. We introduce a ring VRF functionality $\mathcal{F}_{\mathsf{rvrf}}$ satisfying the security properties we want to achieve in rVRF. These properties are informally as follows: *randomness* meaning that an evaluation value is random and independent from the message, ring and the public key, *determinism* meaning that $\mathsf{rVRF.Eval}$ outputs always the same evaluation value, *anonymity* meaning that $\mathsf{rVRF.Sign}$ does not give information about its signer, *unforgeability* meaning that an adversary should not generate a forged signature and *uniqueness* meaning that number of verified evaluation values should not be more than the number of the keys in the public key.

In Figure 1, we give a UC functionality $\mathcal{F}_{\mathsf{rvrf}}$ for ring VRFs. In $\mathcal{F}_{\mathsf{rvrf}}$, we suppress associated data and ring commitment details to make our UC functionality more accessible, meaning our ring commitment is simply the full ring. We introduce more variations of ring VRF functionalities in Appendix B with additional security properties.

We give several important remarks that help elucidate $\mathcal{F}_{\mathsf{rvrf}}$ in Figure 1:

R- 1 Each party is distinguished by unique verification key which is given by the simulator. Verification keys have the identifier role of the signatures and outputs rather than influencing the value of them. Therefore, there exists no secret key as in the real world protocol. We note that we need to define verification keys in $\mathcal{F}_{\mathsf{rvrf}}$ because the real world protocol rVRF defines a verification key (or public key) for each party.

R- 2 In ring VRF, the verification algorithm outputs the corresponding evaluation value of the verified signature. Therefore, $\mathcal{F}_{\mathsf{rvrf}}$ outputs the corresponding output during the signature verification 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 of each signature, then there would be no way that $\mathcal{F}_{\mathsf{rvrf}}$ determines the actual verification key of the signature σ and outputs the evaluation value because σ does not need to be associated with the signer's key or unique. Therefore, $\mathcal{F}_{\mathsf{rvrf}}$ associates an anonymous key for each m and pk so that this key behaves as if it is the verification key of the signature during the verification. Since each anonymous key of an honest party selected randomly and independent from m 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.

R- 3 $\mathcal{F}_{\mathsf{rvrf}}$ does not have a separate signing protocol for malicious parties as honest parties because they can generate it as they want. If they generate a signature, it is added to the $\mathcal{F}_{\mathsf{rvrf}}$'s records as valid or invalid when a party comes for the verification of it. Its validity depends on Sim as it can be seen in the condition C3 in Figure 1.

R- 4 Once Sim obtains an anonymous key of a message m generated for an honest party, we let Sim learn the evaluation of m and W. Sim can do this via malicious ring VRF evaluation i.e., send the message $(\mathsf{eval}, \mathsf{sid}, \mathsf{pk}_i, W, m)$. Here, if W is an anonymous key of m, pk, $\mathcal{F}_{\mathsf{rvrf}}$ returns $\mathtt{evaluations}[m, W]$ even if $\mathsf{pk} \neq \mathsf{pk}_i$. $\mathcal{F}_{\mathsf{rvrf}}$ returns it independent from which verification key given in Sim's message.

R- 5 Once Sim obtains an anonymous key of a message m generated for an honest party, it can learn all valid signatures generated by W for a ring ring and m via malicious requests of signatures.

We construct $\mathcal{F}_{\mathsf{rvrf}}$ so that it achieves the following properties:

Randomness: The evaluation of (m, pk_i), which is $\mathtt{evaluations}[m, W]$ where $\mathtt{anonymous\_key\_map}[m, W] = \mathsf{pk}_i$, is randomly selected independent from (m, pk_i).

Each element in $\mathtt{evaluations}$ are selected randomly by $\mathcal{F}_{\mathsf{rvrf}}$. Therefore, $\mathcal{F}_{\mathsf{rvrf}}$ satisfies randomness. Evaluation of (m, 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 m, W. Therefore, honest evaluations are always random and independent from (m, pk_i). Malicious evaluation of (m, pk_i), where pk_i is not an honest verification key, is generated by first assigning an anonymous key W given by Sim to it and then assigning a random value y to m, W. Since $\mathcal{F}_{\mathsf{rvrf}}$ checks whether W is an anonymous key of another verification key, evaluation of (m, pk_i) is always random. If $\mathcal{F}_{\mathsf{rvrf}}$ did not check this, then the evaluation of m, pk_i would be the same as the evaluation of $m, \mathsf{pk}_j \neq \mathsf{pk}_i$ whose anonymous key is W.

Determinism: Once evaluation of (m, pk_i), which is $\texttt{evaluations}[m, W]$ where $\texttt{anonymous\_key\_map}[m, W] = \mathsf{pk}_i$, is set, it cannot be changed.

$\mathcal{F}_{\mathsf{rvrf}}$ satisfies determinism because it checks whether (m, pk_i) is evaluated before any time that it needs it. The only way for Sim to change the evaluation of (m, pk_i) is by changing the anonymous key of (m, pk_i) but the anonymous key cannot change once it is set i.e., $\mathcal{F}_{\mathsf{rvrf}}$ always checks whether there exists W where $\texttt{anonymous\_key\_map}[m, W] = \mathsf{pk}_i$ whenever it needs the evaluation value.

Anonymity: An honest signature σ of a message m verified by a ring ring does not give any information about its signer except that its key is in pk. We define it more formally.

Definition 9 (Anonymity). *$\mathcal{F}_{\mathsf{rvrf}}$ 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 the environment $\mathcal{Z}$ and a distinguisher $\mathcal{D}$ in the real world. $\mathcal{D}$ accesses a signing oracle $\mathcal{O}_{Sign}$ run by $\mathcal{Z}$ that we define below.

- *Given the input $'\mathsf{keygen}'$, $\mathcal{O}_{Sign}$ creates an honest party in the ideal protocol and obtains its verification key pk, stores pk to a list $\mathcal{K}$ and outputs pk.*
- *Given the input $'(\mathsf{pk}, \mathsf{ring}, \mathsf{input})'$, $\mathcal{O}_{Sign}$ commands the honest party with pk to sign m and output the signature σ and the anonymous key W if $\mathsf{pk} \in \mathsf{ring}$ and $\mathsf{pk} \in \mathcal{K}$. Then $\mathcal{O}_{Sign}$ stores input to a list $\mathsf{signed}[\mathsf{pk}]$ and W to a list $\mathcal{W}$. It outputs (σ, W). Otherwise, it outputs $\perp$.*

At some point of the game, $\mathcal{D}$ sends $(\mathsf{ring}, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{input})$ to $\mathcal{Z}$ where $\mathsf{pk}_0, \mathsf{pk}_1 \in \mathsf{ring}$, $\mathsf{input} \notin \mathsf{signed}[\mathsf{pk}_0]$ and $\mathsf{input} \notin \mathsf{signed}[\mathsf{pk}_1]$. Challenger picks a bit b randomly. Then it gives the input $(\mathsf{pk}_b, \mathsf{ring}, \mathsf{input})$ 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. Otherwise, $\mathcal{Z}$ outputs $\perp$ and $\mathcal{A}$ loses the game. During the game if $\mathcal{D}$ outputs $b' = b$, $\mathcal{D}$ wins the game.

Unforgeability: If an honest party with a public key pk never signs a message m for a ring ring, then no party is able to generate a valid signature of m for ring signed by pk.

We need verify that Sim cannot generate a signature σ that signs a message m for a ring ring by an honest party's key pk. In other words, we need to verify that if $\mathcal{F}_{\mathsf{rvrf}}$ never received a message $(\mathsf{sign}, \mathsf{sid}, \mathsf{ring}, \mathsf{pk}, m)$ from an honest party P with the key pk, $\mathcal{F}_{\mathsf{rvrf}}$ cannot have a record $[m, W, \mathsf{ring}, \sigma, 1]$ such that $\texttt{anonymous\_key\_map}[m, W] = \mathsf{pk}$ (meaning that σ is a valid signature generated by the honest key pk). Sim cannot execute the forgery attack by sending a message $(\mathsf{sign}, \mathsf{sid}, \mathsf{ring}, \mathsf{pk}, m)$ to $\mathcal{F}_{\mathsf{rvrf}}$ because $\mathcal{F}_{\mathsf{rvrf}}$ checks whether the sender of this message is associated with the key pk to generate a signature. Another way for Sim to create a forgery is by sending an honest key $\mathsf{pk}_{\mathsf{Sim}}$ in C3 in Figure 1. However, it is not allowed by $\mathcal{F}_{\mathsf{rvrf}}$ in the condition C3-2 neither.

Uniqueness: The number of verified outputs via signatures for a message and a ring ring is not more than $|\mathsf{ring}|$.

We need to verify that number of outputs for a message m that are verified by the ring ring is not greater than $|\mathsf{ring}|$. Assume that there exist $t + 1$ verified signatures $\mathcal{S} =$

$\{\sigma_1, \sigma_2, \ldots, \sigma_{t+1}\}$ of a message m for a ring ring where $|\mathsf{ring}| = t$ and $\mathcal{F}_{\mathsf{rvrf}}$ outputs $1, y_i$ for each query $(\mathsf{verify}, \mathsf{sid}, \mathsf{ring}, W_i, m, \sigma_i)$ where $y_i \neq y_j$ for all $i \neq j$. Therefore, for each $\sigma_i \in \mathcal{S}$, $\texttt{anonymous\_key\_map}[m, W_i] = \mathsf{pk}_i$ and $\texttt{evaluations}[m, W_i] = y_i$. Clearly, in this case, $W_i \neq W_j$ for all $i \neq j$ since $y_i \neq y_j$ by our assumption. $\mathcal{F}_{\mathsf{rvrf}}$ generates a signature for an honest party if the honest key is in the ring. Also, it generates a unique anonymous key for each pair m, pk. So, if an honest signature for m, ring is verified it means that the honest verification key pk is in ring and (m, pk) has a unique anonymous key so that unique evaluation output. Therefore, the number of anonymous keys (so that evaluation outputs) for the honest signatures is at most number of honest keys in ring. Since we know that once anonymous key W_i is set for m, pk_i, it cannot be changed. This means that there exist at least $|\mathsf{ring}_{mal}| + 1$ signatures in $\mathcal{S}$. When $\mathcal{F}_{\mathsf{rvrf}}$ verifies a malicious signature, it checks in the condition C3-1 how many malicious anonymous keys generated for malicious signatures of m, ring so far. If it is more that $|\mathsf{ring}_{mal}|$, $\mathcal{F}_{\mathsf{rvrf}}$ does not verify it so that it does not output an evaluation value during such signature verification. Therefore, the simulator can generate at most $|\mathsf{ring}_{mal}|$ anonymous keys for verified signatures for ring. This implies that the number of verified outputs of malicious parties is $|\mathsf{ring}_{mal}|$.

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. This happens only when it selects an honest anonymous key which already existed. Since this happens in negligible probability, $\mathcal{F}_{\mathsf{rvrf}}$ is robust.

4 Ring VRF evaluation proofs

We construct ring VRFs with an efficient evaluation proof, which we call the Pedersen VRF and denote PedVRF. PedVRF instantiates the NIZK for the language $L_{\texttt{eval}}$ defined in §1.

An EC VRF like [28,29,17] consists of first a verifiable unique function (VUF) given by a Chaum-Pedersen DLEQ proof between the signer's public key $\mathsf{pk} = \mathsf{sk}\, G$ and the VUF output $\mathsf{preout} = \mathsf{sk} H_{\mathbf{G}}(\mathsf{input})$, after which a PRF evaluation yields a VRF output $\mathsf{out} = H'(\mathsf{input}, h\, \mathsf{preout})$ ala [26, Proposition 1]. Our PedVRF alters the EC VRF by replacing the public key by a Pedersen commitment $\mathsf{pk} + \mathsf{oppk}\, K$ to the secret key sk.

We refer readers to §2.1 for notation, but highlight several points: Although non-essential, we prefer if preout cannot be verified by pairings, as a miss-use resistance measure. We therefore suggest a "sister" Edwards curve $\mathbb{G}'$ with a subgroup $\mathbf{G}'$ of the same order p as $\mathbf{G}_1$, and cofactor h' divisible by 4. You may take $\mathbb{G}' = \mathbb{G}_1$ and $h' = 1$ if reading only for security proofs, but this incurs risks in deployments. As Groth16 dominates our ring VRF verification costs, we describe only the non-batchable PedVRF analogous to [28,29,17], and ignore the cofactor h_1 of $\mathbf{G}_1$ of $\mathbb{G}_1$, but caution [15] impacts batch verifiable flavors. We select a generator G of $\mathbf{G}$ as our public key base point by any desired method, but then fix a second generator K of $\mathbf{G}$ independent from G.

We now describe the Pedersen VRF PedVRF evaluation proof used by our ring VRF: KeyGen works exactly like EC VRF, but Eval differs by not injecting pk into input:

- $\mathsf{PedVRF.KeyGen}$ returns $\mathsf{sk} \leftarrow\!\!\$\, \mathbb{F}_p$ and $\mathsf{pk} = \mathsf{sk}\, G$.
- $\mathsf{PedVRF.Eval} : (\mathsf{sk}, \mathsf{input}) \mapsto H'(\mathsf{input}, h'\, \mathsf{sk}\, H_{\mathbf{G}'}(\mathsf{input}))$

We add two new algorithms that differ from regular VRF constructions. These handle a Pedersen commitment to the secret key sk, together with its opening oppk.

- $\mathsf{PedVRF.CommitKey}(\mathsf{pk})$ returns a blinding factor $\mathsf{oppk} \leftarrow\!\!\$\, \mathbb{F}_p$ and a commitment $\mathsf{compk} = \mathsf{pk} + \mathsf{oppk}\, K$.
- $\mathsf{PedVRF.OpenKey}(\mathsf{compk}, \mathsf{oppk})$ returns $\mathsf{pk} = \mathsf{compk} - \mathsf{oppk}\, K$.

These hide their public key argument pk too, but alone do not bind pk. A successful $\mathsf{PedVRF.Verify}$ below proves knowledge of compk, which makes the commitment binding.

- $\mathsf{PedVRF.Sign} : (\mathsf{sk}, \mathsf{oppk}, \mathsf{input}, \mathsf{ass}) \mapsto \sigma$ First computes $\mathsf{inbase} := H_{\mathbf{G}'}(\mathsf{input})$ and $\mathsf{preout} := \mathsf{sk}\,\mathsf{inbase}$. Next samples $r_1, r_2 \leftarrow\!\$\ \mathbb{F}_p$, computes $R = r_1 G + r_2 K$, and $R_m = r_1 \mathsf{inbase}$, $c = H_p(\mathsf{ass}, \mathsf{input}, \mathsf{compk}, \mathsf{preout}, R, R_m)$, along with $s_1 = r_1 + c\,\mathsf{sk}$ and $s_2 = r_2 + c\,\mathsf{oppk}$. and finally returns the signature $\sigma = (\mathsf{preout}, c, s_1, s_2)$.
- $\mathsf{PedVRF.Verify} : (\mathsf{compk}, \mathsf{input}, \mathsf{ass}, \sigma) \mapsto \mathsf{out} \vee \perp$ parses $\sigma = (\mathsf{preout}, c, s_1, s_2)$, recomputes $\mathsf{inbase} := H_{\mathbf{G}}(\mathsf{input})$, as well as $R = s_1 G + s_2 K - c\,\mathsf{compk}$, and $R_m = s_1 \mathsf{inbase} - c\,\mathsf{preout}$, and finally if $c = H_p(\mathsf{ass}, \mathsf{input}, \mathsf{compk}, \mathsf{preout}, R, R_m)$ then it returns $H'(\mathsf{input}, h'\,\mathsf{preout})$, which equals $\mathsf{PedVRF.Eval}(\mathsf{sk}, \mathsf{input})$, or else returns failure $\perp$ otherwise.

We remark thatPedVRF becomes EC VRF if we demand $\mathsf{oppk} = 0 = r_2$ in Sign. A signature $\sigma = (c, s_1, s_2)$ is a Fiat-Shamir transform of a sigma protocol that shows equality of the discrete logarithm of preout in base inbase and the committed secret key in compk. We nevertheless define security only for our ring VRF constructions, as PedVRF itself exhibits surprising properties.

As described in §1, we instantiate a rVRF-AD from PedVRF plus a ring commitment scheme $\mathsf{rVRF}.\{\mathsf{CommitRing}, \mathsf{OpenRing}\}$. In our construction, $\mathsf{rVRF.Eval} = \mathsf{PedVRF.Eval}$. and $\mathsf{rVRF.KeyGen} = \mathsf{PedVRF.KeyGen}$. We alter $\mathsf{rVRF.KeyGen}$ somewhat in §5.2 by adopting a SNARK friendly public key, although the pk here still exists under the hood. We discuss further nuances of this in §8.2.

As mentioned in §1, the signing algorithm of our ring VRF construction shows that the public key of the committed secret key is in the ring. Therefore, we need a zero-knowledge ring membership proof for the relation $L_{\mathtt{ring}}$ which handles both $\mathsf{PedVRF.OpenKey}$ and $\mathsf{rVRF.OpenKey}$ efficiently.

$$L_{\mathtt{ring}} = \left\{ \mathsf{compk}, \mathsf{comring} \;\middle|\; \exists \mathsf{oppk}, \mathsf{opring} \text{ s.t. } \begin{array}{l} \mathsf{PedVRF.OpenKey}(\mathsf{compk}, \mathsf{oppk}) \\ = \mathsf{rVRF.OpenRing}(\mathsf{comring}, \mathsf{opring}) \end{array} \right\}.$$

- $\mathsf{rVRF.Sign} : (\mathsf{sk}, \mathsf{opring}, \mathsf{input}, \mathsf{ass}) \mapsto \rho$ returns $\rho = (\mathsf{compk}, \pi_{\mathtt{ring}}, \sigma)$ where
  $(\mathsf{oppk}, \mathsf{compk}) \leftarrow \mathsf{PedVRF.CommitKey}$,
  $\pi_{\mathtt{ring}} \leftarrow \mathsf{NIZK}_{L_{\mathtt{ring}}}.\mathsf{Prove}((\mathsf{compk}, \mathsf{comring}); (\mathsf{oppk}, \mathsf{opring}))$,
  $\mathsf{ass}' \leftarrow \mathsf{ass} +\!\!+\, \mathsf{compk} +\!\!+\, \pi_{\mathtt{ring}} +\!\!+\, \mathsf{comring}$,
  $\sigma \leftarrow \mathsf{PedVRF.Sign}(\mathsf{sk}, \mathsf{oppk}, \mathsf{input}, \mathsf{ass}')$.
- $\mathsf{rVRF.Verify} : (\mathsf{comring}, \mathsf{input}, \mathsf{ass}, \rho) \mapsto \mathsf{out} \vee \perp$ parses ρ as $(\mathsf{compk}, \pi_{\mathtt{ring}}, \sigma)$, next sets $\mathsf{ass}' \leftarrow \mathsf{ass} +\!\!+\, \mathsf{compk} +\!\!+\, \pi_{\mathtt{ring}} +\!\!+\, \mathsf{comring}$, aborts if $\mathsf{NIZK}_{L_{\mathtt{ring}}}.\mathsf{Verify}((\mathsf{compk}, \mathsf{comring}); \pi_{\mathtt{ring}})$ fails, and returns $\mathsf{PedVRF.Verify}(\mathsf{compk}, \mathsf{input}, \mathsf{ass}', \sigma)$.

Appendix A proves our ring VRF construction realizes $\mathcal{F}_{\mathsf{rvrf}}$ in Figure 1. 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 comes from the perfect hiding property of Pedersen commitment, the zero-knowledge property of $\mathsf{NIZK}_{L_{\mathtt{ring}}}$ (Lemma 3) and the difficulty of DDH in $\mathbf{G}$ (Lemma 4) 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 5), i.e., for unforgeability, one cannot commit an honest party's secret key without breaking the CDH problem and for the uniqueness, if one can obtain PedVRF signatures such that $\sigma_1 = (\mathsf{preout}_1, \pi_{\mathsf{PedVRF}})$ and $\sigma_2 = (\mathsf{preout}_2, \pi'_{\mathsf{PedVRF}})$ where $\mathsf{preout}_1 \neq \mathsf{preout}_2$ and verified by compk for the message input, then we break a CDH problem in $\mathbf{G}$.

Theorem 1. rVRF *over the group structure* $(\mathbf{G}, p, G, K)$ *realizes* $\mathcal{F}_{\mathsf{rvrf}}$ *in Figure 1 in the random oracle model assuming that NIZK is zero-knowledge and knowledge sound, the decisional Diffie-Hellman (DDH) problem are hard in* $\mathbf{G}$.

5 Zero-knowledge continuations

We now develop our zero-knowledge continuation technique, called *special Groth16* or SpecialG, and build ring VRFs with fast amortized prover time. Anonymity demands we finalize the amortized "continuation" multiple times, with each time being unlinkable to the others, meaning our rVRF stays zero-knowledge even with the continuation being reused.

5.1 Rerandomization

Zero-knowledge invariably comes from random blinding factors. Zero-knowledge continuations need rerandomizable zkSNARKs, meaning Groth16 [19], but beyond rerandomization their unlinkability demands hiding public inputs. In our case, we "specialize" Groth16 to permit alteration of oppk in the PedVRF.OpenKey invocation without reproving our heavy rVRF.OpenRing invocation.

In Groth16 [19], we have an SRS S consisting of curve points in $\mathbf{G}_1$ and $\mathbf{G}_2$ that encode the circuit being proven. We follow [19] in discussing the SRS S in terms of its "toxic waste" $(\alpha, \beta, \delta, \gamma, \tau^1, \tau^2, \ldots) \in \mathbb{F}_q^*$. In other words, we could write say $[f(\tau)/\delta]_1$ or $[\delta]_2$ to denote an element of our SRS S in $\mathbf{G}_1$ or $\mathbf{G}_2$ respectively, computed by scalar multiplication of the Groth16 generators from the toxic waste τ and δ, but for which nobody knows the underlying τ or δ anymore.

In the SRS S, we distinguish the verifiers' string of elements $\chi_1, \ldots, \chi_k, [\alpha]_1 \in \mathbf{G}_1$ and $[\beta]_2, [\gamma]_2, [\delta]_2 \in \mathbf{G}_2$. A Groth16 [19] proof takes the form $\pi = (A, B, C) \in \mathbf{G}_1 \times \mathbf{G}_2 \times \mathbf{G}_1$. A verifier then produces a $X = \sum_i^k x_i \chi_i \in \mathbf{G}_1$ from the public inputs x_i and then checks

$$e(A, B) = e([\alpha]_1, [\beta]_2) \cdot e(X, [\gamma]_2) \cdot e(C, [\delta]_2).$$

We need the rerandomization algorithm from [3, Fig. 1]: An existing SNARK (A, B, C) is transformed into a fresh SNARK (A', B', C') by sampling random $r_1, r_2 \in \mathbb{F}_p$ and computing

$$\begin{aligned} A' &= \frac{1}{r_1} A \\ B' &= r_1 B + r_1 r_2 [\delta]_2 \\ C' &= C + r_2 A. \end{aligned}$$

At this point, our x_i remain identical after rerandomization, so X links (A, B, C) to (A', B', C'). Alone rerandomization cannot alter public inputs x_i, so we instead need an opaque public input point X, which then becomes part of our proof and incurs its own separate proof of correctness.

We build *special Groth16* aka SpecialG by adding one fresh basepoint K_γ independent from all others, including the $H_{\mathbf{G}}(\mathsf{input})$ in PedVRF.[3] In the trusted setup, we build one additional prover SRS element

$$K_\delta := \frac{\gamma}{\delta} K_\gamma.$$

After K_δ is created, our toxic waste γ and δ disappear and subversion resistance could be checked like in [**?**] plus also checking

$$e(K_\gamma, [\gamma]_2) = e(K_\delta, [\delta]_2).$$

We now have a zero-knowledge continuation $\pi = (X, A, B, C)$ from which our algorithm SpecialG.Reprove : $(X, A, B, C) \mapsto ((X', A', B', C'); b)$ produces an unlinkable instance $\pi' =$

[3] Apply the underlying $H_{\mathbf{G}}$ to an input outside the input domain for example.

(X', A', B', C') by first sampling random $b, r_1, r_2 \in \mathbb{F}_p$ and then computing

$$\begin{aligned} X' &= X + bK_\gamma \\ A' &= \frac{1}{r_1} A \\ B' &= r_1 B + r_1 r_2 [\delta]_2 \\ C' &= C + r_2 A + bK_\delta . \end{aligned}$$

As our two b terms cancel in the pairings, our special Groth16 rerandomization reduces to the standard Groth16 rerandomization construction above, except with X now an opaque Pedersen commitment.

Along side opaque inputs in $X = \sum_i^k x_i \chi_i$, our verifier should typically enforce specific values by assembling a few *transparent* inputs $Y = \sum_i^l y_i \Upsilon_i$ themselves. In particular, our ring VRF verifiers should enforce the commitment $\mathsf{comring}$ for ring, even if they outsource computing $\mathsf{comring}$. We thus write $\mathsf{SpecialG.Preprove} : (\bar{y}, \bar{x}; \bar{\omega}) \mapsto (X, A, B, C)$ where $(A, B, C) \leftarrow \mathsf{Groth16.Prove}(\bar{y}, \bar{x}; \bar{\omega})$, so a full Prove algorithm works by composing $\mathsf{Preprove}$ and $\mathsf{Reprove}$.

At this point $\mathsf{SpecialG.Verify}(\bar{y}; (X', A', B', C'))$ computes $X' + Y = X' + \sum_i^l y_i \Upsilon_i$ and checks the tuple $(X' + Y, A', B', C')$ like Groth16 does,

$$e(A', B') = e([\alpha]_1, [\beta]_2) \cdot e(X' + Y, [\gamma]_2) \cdot e(C', [\delta]_2).$$

As our verifier does not build X' themselves, we prove nothing with this pairing equation unless the verifier separately checks a proof-of-knowledge that $X' = bK_\gamma + \sum_i^k x_i \chi_i$ for some unknown $b, \bar{x}$.

Lemma 1. *Our rerandomization procedure transforms honestly generated* $\mathsf{SpecialG}$ *zero-knowledge continuation* (X, A, B, C) *into identically distributed* $\mathsf{SpecialG}$ *proof* (X', A', B', C'), *with identical opaque inputs* $x_1, \ldots, x_k$ *and identical transparent inputs* $y_1, \ldots, y_l$.

Proof (Proof idea.). Adapt the proof of Theorem 3 in [3, Appendix C, pp. 31].

All told, our opaque rerandomization trick converts any conventional Groth16 zkSNARK π for $\mathsf{rVRF.OpenRing}$ into a zkSNARK π' with inputs split into a transparent part $\bar{y}$ vs opaque unlinkable part X.

Importantly, rerandomization requires only four scalar multiplications on $\mathbb{G}_1$ and two scalar multiplications on $\mathbb{G}_2$, which BLS12 curves make roughly equivalent to eight scalar multiplications on $\mathbb{G}_1$.

Lemma 2. *Assuming AGM plus the* $(2n-1, n-1)$*-DLOG assumption, and circuit size less than* n*, then our zero-knowledge continuation* $\mathsf{SpecialG}$ *satisfies knowledge soundness.*

Proof (Proof idea.). As our Prove is composed from $\mathsf{Preprove}$ and $\mathsf{Reprove}$, our challenger learns the actual public input wire values and blinding factors. Adapt the proof of Theorem 2 in [3, §3, pp. 9], observing that K_γ and K_δ never interact with other elements.

In fact, one could prove zero-knowledge continuations satisfy weak white-box simulation extractability, much like Theorem 1 in [3, §3, pp. 8 & 11]. We depend upon the specific simulator though, which itself increases our dependence upon the usage of the zero knowledge continuation.

5.2 Continuation

We describe a much faster choice $\pi_{\texttt{fast}}$ for $\pi_{\texttt{ring}}$ that sets $x_1 = \mathsf{sk}$ and $x_0 = \mathsf{comring}$ so that taking $G = \chi_1$, $K = K_\gamma$, and $\mathsf{oppk} = b$ in PedVRF yields an incredibly fast amortized ring VRF prover. Also PedVRF itself proves knowledge of $X' = \mathsf{sk}\,\chi_1 + bK_\gamma$, as required by SpecialG.Verify.

A priori, we do not know χ_1 during the trusted setup for $\pi_{\texttt{fast}}$, which prevents computing $\mathsf{pk} = \mathsf{sk}\,\chi_1$ inside $\pi_{\texttt{fast}}$.[4] Instead, we propose ring contain commitments to sk over some Jubjub curve $\mathbb{J}$.

We know the large subgroup $\mathbf{J}$ of $\mathbb{J}$ typically has smaller prime order $p_{\mathbf{J}}$ than $\mathbf{G}$, itself due to $\mathbb{J}$ being an Edwards curve. We thus choose $\mathsf{sk}_0, \mathsf{sk}_1 < p_{\mathbf{J}}$ so that $\mathsf{PedVRF.sk} = \mathsf{sk}_0 + \mathsf{sk}_1\, 2^{128} \bmod p_{\mathbf{G}}$, and so our rVRF.KeyGen returns a secret key of the form $\mathsf{rVRF.sk} = (\mathsf{sk}_0, \mathsf{sk}_1, d)$ with $d \leftarrow\!\!\$\; \mathbb{F}_{p_{\mathbf{J}}}$ and a public key of the form $\mathsf{rVRF.pk} = \mathsf{sk}_0\, J_0 + \mathsf{sk}_1\, J_1 + dJ_2$, for some independent J_0, J_1, J_2.[5]

$$L^{\texttt{inner}}_{\texttt{fast}} = \left\{ \mathsf{sk}_0 + 2^{128}\mathsf{sk}_1, \mathsf{comring} \;\middle|\; \exists d, \mathsf{opring} \text{ s.t. } \begin{array}{c} \mathsf{rVRF.OpenRing}(\mathsf{comring}, \mathsf{opring}) \\ = \mathsf{sk}_0 J_0 + \mathsf{sk}_1 J_1 + dJ_2 \end{array} \right\}$$

Applying our rerandomization Reprove to $\pi^{\texttt{inner}}_{\texttt{fast}}$ with opaque input yields a zkSNARK $\pi_{\texttt{fast}}$ with the extra PedVRF.OpenKey arithmetic to have exactly the form $\pi_{\texttt{ring}}$.

We explain later in §7 how one could choose χ_1 independent before doing the trusted setup, and then wire χ_1 into $\pi_{\texttt{fast}}$ inside C. In this case, we could prove $\mathsf{pk} = \mathsf{sk}\,\chi_1$ inside $\pi_{\texttt{fast}}$, but then non-native arithmetic makes $\pi_{\texttt{fast}}$ far slower.

At this point, PedVRF.Sign requires two scalar multiplications on $\mathbb{G}_1$ and two on $\mathbb{G}'$, so together with rerandomization costing four scalar multiplications on $\mathbb{G}_1$ and two on $\mathbb{G}_2$, our amortized prover time comes under 12 scalar multiplications on typical $\mathbb{G}_1$ curves. We expect the three pairings dominate verifier time, but verifiers also need five scalar multiplications on $\mathbb{G}_1$.

As an aside, one could construct a second faster curve with the same group order as $\mathbf{G}$, which speeds up two scalar multiplications in both the prover and verifier.

Importantly, our fast ring VRF' amortized prover time now rivals group signature schemes' performance. We hope this ends the temptation to deploy group signature like constructions where the deanonymization vectors matter.

Theorem 2. rVRF *instantiated with* $\pi_{\texttt{fast}}$ *and* PedVRF *satisfies knowledge soundness.*

Proof (Proof stetch.). An extractor for PedVRF reveals the opening of X for us, so our result follows from Lemma 2.

6 Ring updates

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

[4] We challenge this statement in §7 below, but do not suggest those technique here, give now slow non-native field arithmetic is.

[5] Interestingly we avoid range proofs for sk_1 and sk_2 by this independence.

6.1 Merkle trees

Our $\mathsf{rVRF}.\{\mathsf{CommitRing}, \mathsf{OpenRing}\}$ could implement a Merkle tree using a zkSNARK friendly hash function like Poseidon [18], giving $O(\log|\mathsf{ring}|)$ prover time. At least one Poseidon [18] 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.

6.2 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 $\pi_{\mathtt{sk}}$ and a Groth16 or KZG π_{pk} for two respective languages:

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

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

We now prove $\pi_{\mathtt{sk}}$ 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 $\pi_{\mathtt{sk}}$ 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 $\pi_{\mathtt{sk}}$ bridges between the PedVRF and π_{pk}, one might consider the $\pi_{\mathtt{sk}}$-to-π_{pk} continuation to be "time reversed", in that the "middle" continuation is proved first.

6.3 Polynomial commitments

As π_{pk} became rather simple, there exists an alternative formulation: $\mathsf{comring}$ could be a KZG polynomial commitment [23] to users' J_{pk}s, while π_{pk} itself becomes an opening at a secret location, like Caulk+ [31] or Caulk [34]. We benefit from faster ring updates this way, but pay in increased verifier time and increased marginal prover time.

6.4 Append only rings

As a slight variation, we could build ring using append only structures like some blockchains, in which case we should split $\mathsf{rVRF}.\mathsf{OpenRing}$ differently between an inner ring block or epoch proof $L_{\mathtt{block}}$, which we only prove once like $\pi_{\mathtt{sk}}$ above, and a chain state proof $L_{\mathtt{chain}}$, 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.

$$L_{\mathtt{chain}}^{\mathtt{inner}} = \{\, \mathtt{blk}, \mathtt{chain} \mid \mathtt{blk} \in \mathtt{chain} \,\}\,, \quad \text{and}$$

$$L_{\mathtt{block}}^{\mathtt{inner}} = \left\{ \mathsf{sk}_0 + \mathsf{sk}_1 2^{128}, \mathtt{blk} \;\middle|\; \exists d, \mathsf{opring} \text{ s.t. } \begin{array}{c} \mathsf{OpenRing}(\mathtt{blk}, \mathsf{opring}) \\ = \mathsf{sk}_0 J_0 + \mathsf{sk}_1 J_1 + dJ_2 \end{array} \right\}.$$

We suggest appending `blk` to a polynomial commitment using [33], which then $L_{\mathtt{chain}}$ blind opens via Caulk+ [31] as above.

6.5 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 $\pi_{\mathtt{sk}}$ or $L_{\mathtt{block}}$ commit to the expiration date alongside the secret key in their X, and then $\pi_{\mathtt{pk}}$ or $L_{\mathtt{chain}}$ enforce expiration, but really even PedVRF could enforce expiration.

A revocation list could be enforced by a non-membership proof in $\pi_{\mathtt{pk}}$ or $L_{\mathtt{chain}}$. 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.

7 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.

7.1 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 §6.2. We could then blind open a polynomial commitment [23] to our comring choices, Caulk+ [31] or Caulk [34] or similar as in §6.3.

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 comring choices across the history of the same evolving ring database.

7.2 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

$$\chi_1 = \left[\frac{\beta u_1(\tau) + \alpha v_1(\tau) + w_1(\tau)}{\gamma}\right]_1 .$$

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

$$\chi_{1,\delta} := \left[\frac{\beta u_1(\tau) + \alpha v_1(\tau) + w_1(\tau) - \gamma \chi_{1,\gamma}}{\delta}\right]_1 .$$

At this point, we replace χ_1 by $\chi_{1,\gamma}$ everywhere and our proofs add 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 $[\delta]_2, [\gamma]_2 \in \mathbf{G}_2$ and $e([\alpha]_1, [\beta]_2) \in \mathbf{G}_T$. We leave this as an exercise to the reader.

8 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 pseudo-nonymous 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 $\pi_{\mathtt{sk}}$ and $\pi_{\mathtt{pk}}$ 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 $\pi_{\mathtt{fast}}$ every time ring changes, likely 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 ass. Our user agent computes the user's identity $\mathtt{id} = \mathsf{PedVRF.Eval}(\mathsf{sk}, \mathsf{input})$ on the server input, and finally sends the server their ring VRF signature $\mathsf{rVRF.Sign}(\mathsf{sk}, \mathsf{opring}, \mathsf{input}, \mathsf{ass})$
- *Verify* – After receiving $(\mathsf{compk}, \pi_{\mathtt{fast}}, \sigma)$ in channel ass, the server named input checks $\pi_{\mathtt{fast}}$ on the input $\mathsf{compk} + \mathsf{comring}$, checks the VRF signature and obtains the user's identity $\mathtt{id}$, ala
$\mathtt{id} = \mathsf{PedVRF.Verify}(\mathsf{compk}, \mathsf{input}, \mathsf{ass} + \!\!\!+ \mathsf{compk} + \!\!\!+ \pi_{\mathtt{fast}}, \sigma)$.

8.1 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.

8.2 AML/KYC

We shall not discuss AML/KYC in detail, because the entire field lacks clear goals, and thus winds up being ineffective [30]. 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 §5.2 and §6.2 costs us exculpability from Definition 10.

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

8.3 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 [14], we merely delinearize inbase and preout in Sign and Verify like:

$$\begin{aligned} x &= H(\mathsf{input}_j, \mathtt{id}_j, \ldots, \mathsf{input}_j, \mathtt{id}_j) \\ \mathsf{inbase} &= \sum_j H_p(x, j)\, \mathsf{inbase}_j \\ \mathsf{preout} &= \sum_j H_p(x, j)\, \mathsf{preout}_j \end{aligned}$$

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 $\mathsf{input}_1 = \mathtt{site\_name} \mathbin{+\!\!+} \mathtt{current\_month}$ and $\mathsf{input}_2 = \mathtt{site\_name} \mathbin{+\!\!+} \mathtt{registration\_month}$, 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 §8.2 too.

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

8.4 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 §5 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 $(\mathtt{id}, H(\mathsf{compk} \mathbin{+\!\!+} \pi_{\mathtt{fast}}), \mathtt{dates})$ after their first verification, but then skipping the Groth16 check after merely rechecking the hash $H(\mathsf{compk} \mathbin{+\!\!+} \pi_{\mathtt{fast}})$.

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 $H(\mathsf{compk} \mathbin{+\!\!+} \pi_{\mathtt{fast}})$ changed. We rate limit or verify more lazily users with many nearby login dates

9 Application: Rate limiting

We showed in §8 how ring VRFs give users only one unique identity for each input input. We explained in §8.3 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 $\mathsf{input} = \mathtt{domain} \mathbin{+\!\!+} \mathtt{date} \mathbin{+\!\!+} k$.

9.1 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 $\mathsf{input}_i = \mathtt{domain} \mathbin{+\!\!+} \mathtt{date} \mathbin{+\!\!+} k_i$ 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 §7.

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 §1. Although interesting, we caution these bring performance concerns not discussed here, so deployments should consider if leaking k suffices.

9.2 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 [25]. At these levels, we experience devastating famines as the Earth's carrying capacity drops below one billion people [32]. 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 [11]. 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.

9.3 Multi-constraint rationing

As in §8.3, 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.

9.4 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 §8.4 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 [6] or PrivacyPass OPRF tokens [14], 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.

9.5 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 [6] 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 ass 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. Christian Badertscher, Peter Gaži, Iñigo Querejeta-Azurmendi, and Alexander Russell. On uc-secure range extension and batch verification for ecvrf. *Cryptology ePrint Archive*, 2022.
3. 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.
4. Adam Bender, Jonathan Katz, and Ruggero Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, pages 60–79, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. https://ia.cr/2005/304.
5. 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.
6. Jeffrey Burdges, Florian Dold, and Christian Grothoff. Robust and income-transparent online e-cash.
7. Privacy by Design Foundation. Irma docs v0.2.0. https://irma.app/docs/v0.2.0/overview/.
8. 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.
9. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. https://eprint.iacr.org/2000/067.
10. 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.
11. Chatham House. Climate change risk assessment 2021.
12. 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.

13. 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.
14. 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.
15. Henry de Valence. It's 255:19am. do you know what your validation criteria are? `https://hdevalence.ca/blog/2020-10-04-its-25519am`.
16. 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.
17. 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.
18. 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`.
19. 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.
20. Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *Journal of the ACM (JACM)*, 59(3):1–35, 2012.
21. Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Aggregatable distributed key generation. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021*, volume 12696 of *LNCS*, pages 147–176, 2021. `https://ia.cr/2021/005`.
22. 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.
23. 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.
24. 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`.
25. 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.
26. 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.
27. 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/`.
28. 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`.
29. Trevor Perrin. The xeddsa and vxeddsa signature schemes. Revision 1, 2016-10-20. `https://signal.org/docs/specifications/xeddsa/`.
30. 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.
31. Jim Posen and Assimakis A. Kattis. Caulk+: Table-independent lookup arguments. Cryptology ePrint Archive, Paper 2022/957, 2022. `https://eprint.iacr.org/2022/957`.
32. David Spratt. At 4°c of warming, would a billion people survive? what scientists say.
33. 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`.
34. 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

Before giving the security proof of our protocol, we give the protocol in Section 4 without the abstraction from PedVRF for the sake of clarity of the security proof.

We instantiate parameter generation by constructing a group $\mathbf{G}$ of order p and two generators $G, K \in \mathbf{G}$. We consider three hash functions: $H, H' : \{0,1\}^* \to \mathbb{F}_p$ and a hash-to-group function $H_{\mathbf{G}} : \{0,1\}^* \to \mathbf{G}$ and . rVRF works as follows:

- $\mathsf{rVRF.KeyGen}(1^\kappa)$: It selects randomly a secret key $x \in \mathbb{F}_p$ and computes the public key $X = xG$. In the end, it outputs $\mathsf{sk} = x$ and $\mathsf{pk} = X$.
- $\mathsf{rVRF.Sign}(\mathsf{sk}, \mathsf{ring}, m)$: It lets $\mathsf{inbase} = H_{\mathbf{G}}(m)$ and computes the pre-output $\mathsf{preout} = x\mathsf{inbase}$. The signing algorithm works as follows:
  - It first commits to its secret key x i.e., $\mathsf{compk} = X + \mathsf{oppk}\, K$ where $\mathsf{oppk} \leftarrow\!\$\ \mathbb{F}_p$.
  - It generates a Chaum-Pedersen DLEQ proof π_{dleq} showing the following relation by running the algorithm $\mathsf{NIZK}_{R_{\mathsf{dleq}}}.\mathsf{Prove}(((G, K, \mathbf{G}, \mathsf{compk}, \mathsf{preout}, \mathsf{inbase}); (x, \mathsf{oppk})))$ which outputs $\to \pi_{dleq}$

    $$R_{\mathsf{dleq}} = \{((x, \mathsf{oppk}), (G, K, \mathbf{G}, \mathsf{compk}, \mathsf{preout}, \mathsf{inbase})) : \mathsf{compk} = xG + \mathsf{oppk}\, K, \mathsf{preout} = x\, \mathsf{inbase}\} \quad (1)$$

    Here Prove algorithm runs a non-interactive Chaum-Pedersen DLEQ proof with the Fiat-Shamir transform: Sample random $r_1, r_2 \leftarrow \mathbb{F}_p$. Let $R = r_1 G + r_2 K$, $R_m = r_1 \mathsf{inbase}$, and $c = H'(\mathsf{ring}, m, \mathsf{preout}, \mathsf{compk}, R, R_m)$. Set $\pi_{dleq} = (c, s_1, s_2)$ where $s_1 = r_1 + cx$ and $s_2 = r_2 + c\, \mathsf{oppk}$.
  - It generates the second proof π_{ring} for the following relation with the witness $(\mathsf{ring}, x, \mathsf{oppk})$.

    $$\mathcal{R}_{\mathsf{ring}} = \{(X, \mathsf{oppk}), (G, K, \mathbf{G}, \mathsf{ring}, \mathsf{compk})) : \mathsf{compk} - \mathsf{oppk}\, K = X \in \mathsf{ring}\} \quad (2)$$

    The second proof π_{ring} is generated by running $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{ring}}}.\mathsf{Prove}(((G, K, \mathbf{G}, \mathsf{ring}, \mathsf{compk}); (X, \mathsf{oppk})))$

  In the end, $\mathsf{rVRF.Sign}$ outputs $\sigma = (\pi_{dleq}, \pi_{ring}, \mathsf{compk}, \mathsf{preout})$.
- $\mathsf{rVRF.Verify}(\mathsf{ring}, \mathsf{preout}, m, \sigma)$: Given $\sigma = (\pi_{dleq}, \pi_{ring}, \mathsf{compk})$ and $\mathsf{ring}, \mathsf{preout}$, it runs $\mathsf{NIZK}_{R_{\mathsf{dleq}}}.\mathsf{Verify}((G, K, \mathbf{G}, \mathsf{compk}, \mathsf{preout}, \mathsf{inbase}), \pi_{dleq})$ where $P = H_{\mathbf{G}}(m)$. $\mathsf{NIZK}_{R_{\mathsf{dleq}}}.\mathsf{Verify}$ works as follows: $\pi_{dleq} = (c, s_1, s_2)$, it lets $R' = s_1 G + s_2 K - c\, \mathsf{compk}$ and $R'_m = s_1 H_{\mathbf{G}}(m) - c\, \mathsf{preout}$. It returns true if $c = H'(\mathsf{ring}, m, \mathsf{preout}, \mathsf{compk}, R', R'_m)$. If $\mathsf{NIZK}_{R_{\mathsf{dleq}}}.\mathsf{Verify}((G, K, \mathbf{G}, \mathsf{compk}, \mathsf{preout}, \mathsf{inbase}), \pi_{dleq})$ outputs 1, it runs $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{ring}}}.\mathsf{Verify}((G, K, \mathbf{G}, \mathsf{ring}, \mathsf{compk}), \pi_{ring})$. If all verification algorithms verify, it outputs 1 and the evaluation value $y = H(m, \mathsf{preout})$. Otherwise, it outputs $(0, \perp)$.

A.1 Security Analysis

Before we start to analyse our protocol, we should define the algorithm Gen_{sign} for $\mathcal{F}_{\mathsf{rvrf}}$ and show that $\mathcal{F}_{\mathsf{vrf}}$ with Gen_{sign} satisfies the anonymity defined in Definition 9. $\mathcal{F}_{\mathsf{rvrf}}$ that rVRF realizes runs Algorithm 1 to generate honest signatures.

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

1: $c, s_1, s_2 \leftarrow\!\$\ \mathbb{F}_p$
2: $\mathsf{oppk} \leftarrow\!\$\ \mathbb{F}_p$
3: $\mathsf{compk} = \mathsf{pk} + \mathsf{oppk}\, K$
4: $\pi_{dleq} \leftarrow (c, s_1, s_2)$
5: $\pi_{ring} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{\mathsf{ring}}}.\mathsf{Prove}(((G, K, \mathbf{G}, \mathsf{ring}, \mathsf{compk}); (X, \mathsf{oppk})))$
6: **return** $\sigma = (\pi_{dleq}, \pi_{ring}, \mathsf{compk}, W)$

Lemma 3. *$\mathcal{F}_{\mathsf{rvrf}}$ running Algorithm 1 satisfies anonymity defined in Definition 9 assuming that $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{ring}}}$ is zero-knowledge and Pedersen commitment is perfectly hiding.*

Proof. We run a simulated copy of $\mathcal{Z}$ and 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 $\pi_{ring} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{\mathsf{ring}}}.\mathsf{Simulate}((G, K, \mathbf{G}, \mathsf{ring}, \mathsf{compk})))$. 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 3. *Assuming that $H_{\mathbf{G}}, H, H'$ are random oracles, the DDH problem is hard in the group structure $(\mathbf{G}, G, K, p)$ and NIZK algorithms are zero-knowledge and knowledge sound, rVRF UC-realizes $\mathcal{F}_{\mathsf{rvrf}}$ running Algorithm 1 according to Definition 4.*

Proof. We construct a simulator Sim that simulates the honest parties in the execution of rVRF 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 samples $x \leftarrow\$ \mathbb{F}_p$ and obtains the key $X = xG$. It adds xG to lists `honest_keys` and `verification_keys` as a key of P_i. In the end, Sim returns $(\mathsf{verificationkey}, \mathsf{sid}, X)$ to $\mathcal{F}_{\mathsf{rvrf}}$.
- [**Simulation of corruption:**] Upon receiving a message $(\mathsf{corrupted}, \mathsf{sid}, \mathsf{P}_i)$ from $\mathcal{F}_{\mathsf{rvrf}}$, Sim removes the public key X from `honest_keys` which is stored as a key of P_i and adds X to `malicious_keys`.
- [**Simulation of the random oracles:**] We describe how Sim simulates the random oracles $H_{\mathbf{G}}, H, H'$ 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

<u>**Oracle** $H_{\mathbf{G}}$</u>
Input: m
if `oracle_queries_gg`$[m] = \perp$
 $h \leftarrow\$ \mathbb{F}_p$
 $P \leftarrow hG$
 `oracle_queries_gg`$[m] := h$
else:
 $h \leftarrow$ `oracle_queries_gg`$[m]$
 $P \leftarrow hG$
return inbase

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

  the random oracle H is less straightforward (See Figure 3). The value W can be a pre-output generated by $\mathsf{rVRF.Eval}$ 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. Sim pretends W as if it is a pre-output. So, Sim first

obtains the discrete logarithm h of $H_{\mathbf{G}}(m)$ from the $H_{\mathbf{G}}$'s database and finds out a public key $X^* = h^{-1}W$. If X^* is not an honest key generated by Sim, Sim can obtain $\mathtt{evaluations}[m, W]$ by sending a message $(\mathsf{eval}, \mathsf{sid}, X^*, W, m)$. Otherwise, it replies by a randomly selected value from $\mathbb{F}_p$. Remark that if W is a pre-output generated by $\mathcal{A}$, then $\mathcal{F}_{\mathsf{rvrf}}$ matches it with m, X^* and registers X^* as a malicious key. If W is an anonymous key of an honest party in the ideal world, $\mathcal{F}_{\mathsf{rvrf}}$ does not care X^* and return an honest evaluation value $\mathtt{evaluations}[m, W]$. During the simulation of H, if $\mathcal{F}_{\mathsf{rvrf}}$ aborts, it means that there exists $W' \neq W$ such that $\mathtt{anonymous\_key\_map}[m, W'] = X^*$. This means that $hX^* = W' \neq W$, but it is impossible because $W = hX^*$. Therefore, Sim never aborts during the simulation of H.
We note that $\mathcal{F}_{\mathsf{rvrf}}$ never generates an anonymous key with an honest verification key. 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.

<u>Oracle H</u>
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 verification key
if $X^* \notin \mathtt{honest\_keys}$
  send $(\mathsf{eval}, \mathsf{sid}, W, X^*, m)$ **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' (See Figure 4) checks whether the random oracle query $(\mathsf{ring}, m, W, \mathsf{compk}, R, R_m)$ is an R_{dleq} 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 R_{dleq} verification query. If it is the case, it assigns c as an answer of $H'(\mathsf{ring}, m, W, \mathsf{compk}, R, R_m)$ so that R_{dleq} 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., $\mathtt{oracle\_queries\_h}[m, W] \neq \mathtt{evaluations}[m, W]$. Therefore, Sim aborts if it is ever happen.

– [**Simulation of** verify] Upon receiving $(\mathsf{verify}, \mathsf{sid}, \mathsf{ring}, W, m, \sigma)$ from the functionality $\mathcal{F}_{\mathsf{rvrf}}$, Sim runs the two NIZK verification algorithms run for $R_{\mathsf{dleq}}, \mathcal{R}_{\mathsf{ring}}$ with the input $\mathsf{ring}, m, \sigma, W$

Oracle H'
Input: $(\mathsf{ring}, m, W, \mathsf{compk}, R, R_m)$

send $(\mathsf{request\_signatures}, \mathsf{sid}, \mathsf{ring}, W, m)$
receive $(\mathsf{signatures}, \mathsf{sid}, \mathsf{ring}, m, \mathcal{L}_\sigma)$
if $\exists \sigma \in \mathcal{L}_\sigma$ where $\mathsf{compk} \in \sigma$
  get $\pi_1 = (c, s_1, s_2) \in \sigma$
  $R := s_1 G + s_2 K - c\mathsf{compk}, R_m = s_1 H_{\mathbf{G}}(m, \mathsf{ring}) - cW$
  $h := \texttt{oracle\_queries\_gg}[m, W]$
  if $\texttt{oracle\_queries\_h\_CP}[\mathsf{ring}, m, W, \mathsf{compk}, R, R_m] = \perp$
    $\texttt{oracle\_queries\_h\_CP}[\mathsf{ring}, m, W, \mathsf{compk}, R, R_m] := c$
  else if $(\texttt{oracle\_queries\_h\_CP}[\mathsf{ring}, m, W, \mathsf{compk}, R, R_m] \neq c$
  or $X^* = h^{-1}W \in \texttt{honest\_keys})$: Abort
else:
  $c \leftarrow\$ \mathbb{F}_p$
  $\texttt{oracle\_queries\_h\_CP}[\mathsf{ring}, m, W, \mathsf{compk}, R, R_m] := c$
return $\texttt{oracle\_queries\_h\_CP}[\mathsf{ring}, m, W, \mathsf{compk}, R, R_m]$

Fig. 4. The random oracle H'

described in rVRF.Verify algorithm of rVRF. If all verify, 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 $h = \texttt{oracle\_queries\_gg}[m]$ and sends $(\mathsf{verified}, \mathsf{sid}, \mathsf{ring}, W, m, \sigma, b_{\mathsf{Sim}}, X)$ to $\mathcal{F}_{\mathsf{rvrf}}$ and receives back $(\mathsf{verified}, \mathsf{sid}, \mathsf{ring}, W, m, \sigma, y, b)$.
  - 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 1, 2 or 3. If $\mathcal{F}_{\mathsf{rvrf}}$ is in 1, it means that $\mathsf{counter}[m, \mathsf{ring}] > |\mathsf{ring}_m|$. If $\mathcal{F}_{\mathsf{rvrf}}$ is in 2, it means that X belongs to an honest party but this honest party never signs m for the ring ring. So, σ is a forgery. If $\mathcal{F}_{\mathsf{rvrf}}$ is in 3, it means that there exists $W' \neq W$ where $\texttt{anonymous\_key\_map}[m, W'] = X$. 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 $X = \perp$ and sends $(\mathsf{verified}, \mathsf{sid}, \mathsf{ring}, W, m, \sigma, b_{\mathsf{Sim}}, X)$ to $\mathcal{F}_{\mathsf{rvrf}}$. Then, Sim receives back $(\mathsf{verified}, \mathsf{sid}, \mathsf{ring}, W, m, \sigma, \perp, 0)$.

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 4. *Assuming that 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. Clearly, 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 is generated differently in Algorithm 1 than rVRF.Sign. The distribution of $\pi_{dleq} = (c, s_1, s_2)$ and compk generated by Algorithm 1 and the distribution of $\pi_{dleq} = (c, s_1, s_2)$ and compk generated by rVRF.Sign are from uniform distribution so they are indistinguishable.

Now, we show that the anonymous key W selected randomly fro $\mathbf{G}$ and pre-output$W$ generated by rVRF.Sign are indistinguishable. For this, we need show that selecting W randomly from

$\mathbf{G}$ and computing W as $xH_{\mathbf{G}}(m, \mathsf{ring})$ are indistinguishable. We show this under the assumption that the DDH problem is hard. In other words, we show that if there exists an adversary $\mathcal{A}'$ that distinguishes anonymous keys of honest parties in the ideal world and anonymous key of the honest parties 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 anonymous keys of first i honest parties are computed as described in $\mathsf{rVRF.Sign}$ and the rest are computed by selecting them randomly. 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 anonymous keys of the ideal protocol and H_{n_h} is equivalent to the anonymous keys of honest parties in the real world. We construct an adversary $\mathcal{B}$ that breaks the DDH problem given that there exists an adversary $\mathcal{A}'$ 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 $\mathcal{A}'$ as follows: $\mathcal{B}$ generates the public key of all honest parties' key as usual by running $\mathsf{rVRF.KeyGen}$ except party P_{i+1}. It lets P_{i+1}'s public key be X. $\mathcal{B}$ gives $\mathbf{G}, G = Y, K$ as parameters of rVRF.

$\mathcal{B}$ simulates the ring signatures of first i parties as in the real protocol and the parties $\mathsf{P}_{i+2}, \ldots, \mathsf{P}_{n_h}$ as follows: it generates a ring signature and its anonymous key by running Algorithm 1 and selecting randomly, respectively. The simulation of P_{i+1} is different. It lets the public key of P_{i+1} be X. Whenever P_{i+1} needs to sign an input m for a ring ring where $\mathsf{pk}_{i+1} \in \mathsf{ring}$, it obtains $P = H_{\mathbf{G}}(m) = hY$ from $\texttt{oracle\_queries\_gg}$ and lets $W = hZ$. 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 $\mathcal{A}$ 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 $\mathcal{A}'$ which distinguishes H_i and H_{i+1}. Since DDH problem is hard, Sim has negligible advantage in the DDH game. So, $\mathcal{A}'$ has a negligible advantage too. Hence, from the hybrid argument, we can conclude that H_0 which corresponds the output of honest parties in rVRF 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 5. *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' is a random oracle and* $\mathsf{NIZK}_{R_{\mathsf{dleq}}}$ *and* $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{ring}}}$ *are knowledge sound.*

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 have already shown in Lemma 4 that the output of honest parties are indistinguishable from the real protocol. Next, 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' 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' if $W = hX$ where X is an honest verification key or if $\texttt{oracle\_queries\_h\_CP}[\mathsf{ring}, m, W, \mathsf{compk}, R, R_m]$ has already been defined by a value which is different than c. The first case happens in H' if $\mathcal{F}_{\mathsf{rvrf}}$ selects a random $W \in \mathbf{G}$ for an anonymous key of m and $\mathsf{pk} = X$ and the random oracle $H_{\mathbf{G}}$ selects a random $h \in \mathbb{F}_p$ where $H_{\mathbf{G}}(m) = hG$. Clearly, this can happen with a negligible probability in λ. The second case happens in H' if $\mathcal{A}$ queries with the input $(\mathsf{ring}, m, W, \mathsf{compk}, R, R_m)$ before $(\pi_1, \pi_2, \mathsf{compk}, W)$

generated by Gen_{sign}. Since compk is random, the probability that $\mathcal{A}$ guesses even 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 the CDH problem.

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 with a public key X, it behaves exactly as $\mathcal{F}_{\mathsf{rvrf}}$ except that it runs Algorithm 2 to generate the signature.

Algorithm 2 $\mathsf{Gen}_{sign}(\mathsf{ring}, W, X, m)$

$c, s_1, s_2 \leftarrow\!\!\$\ \mathbb{F}_p$
$\mathsf{oppk} \leftarrow\!\!\$\ \mathbb{F}_p$
$\mathsf{compk} = X + \mathsf{oppk}\, K$
$R' = sG + \delta K + c\mathsf{compk}$
$R_m = sH_{\mathbf{G}}(m, \mathsf{ring}) + cW$
$\mathtt{oracle\_queries\_h\_CP}[\mathsf{ring}, m, W, \mathsf{compk}, R', R'_m] = c$
$\pi_{dleq} \leftarrow (c, s_1, s_2)$
$\pi_{\mathsf{pk}} \leftarrow \mathsf{NIZK}_{\mathcal{R}_{\mathsf{ring}}}.\mathsf{Prove}(((X, \mathsf{oppk}), (G, K, \mathbf{G}, \mathsf{ring}, \mathsf{compk})))$
return $\sigma = (\pi_{dleq}, \pi_{ring}, \mathsf{compk}, W)$

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 the same except that it sets up the random oracle H' so that π_1 verifies for R_{dleq}. Therefore, the simulation of H' 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 generates the public key of each honest party as $r_x V$. 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. Therefore, generating the honest public keys in this way is indistinguishable.

Oracle H
Input: m, W
if $\mathtt{oracle\_queries\_h}[m, W] = \perp$
  $y \leftarrow\!\!\$\ \{0,1\}^{\ell_{\mathsf{rVRF}}}$
  $\mathtt{oracle\_queries\_h}[m, W] := y$
return $\mathtt{oracle\_queries\_h}[m, W]$

Fig. 5. The random oracle H

Simulation of $H_{\mathbf{G}}$ is as described in Figure 6 i.e., it returns hU instead of hG. The simulation of $H_{\mathbf{G}}$ is indistinguishable from the simulation of $H_{\mathbf{G}}$ in Figure 2. $\mathcal{B}$ simulates the random oracle H in Figure 5 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 $\mathcal{F}_{\mathsf{rvrf}}$. This difference is indistinguishable from the simulation of H by Sim because Sim gets it from $\mathcal{F}_{\mathsf{rvrf}}$ which selects it randomly as $\mathcal{B}$ does. Remark that since $H_{\mathbf{G}}$ is not simulated as in Figure 2, $\mathcal{B}$ cannot check whether W is an anonymous key generated by an honest key or not. However, it does not need this information because H is simulated as a usual random oracle.

Oracle $H_{\mathbf{G}}$
Input: m
if `oracle_queries_gg`$[m] = \perp$
  $h \leftarrow\$ \mathbb{F}_p$
  $P \leftarrow hU$
  `oracle_queries_gg`$[m] := h$
else:
  $h \leftarrow$ `oracle_queries_gg`$[m]$
  $P \leftarrow hU$
return inbase

Fig. 6. The random oracle $H_{\mathbf{G}}$

During the simulation whenever a valid signature $\sigma = (\pi_{dleq}, \pi_{\mathsf{pk}}, \mathsf{compk}, W)$ of message m signed by ring is outputted and $W \notin \mathcal{W}[m, \mathsf{ring}]$, Sim increments a counter $\mathsf{counter}[m, \mathsf{ring}]$ and adds W to $\mathcal{W}[m, \mathsf{ring}]$. Then it runs the extractor algorithm of $\mathsf{NIZK}_{\mathcal{R}_{\mathsf{ring}}}$ i.e., $\mathsf{Ext}(\mathcal{R}_{\mathsf{ring}}, \pi_{\mathsf{pk}_j}, (G, K, \mathbf{G}, \mathsf{ring}, \mathsf{compk})) \to X, \mathsf{oppk}$ where $X \in \mathsf{ring}$ and $\mathsf{compk} = X + \mathsf{oppk}\, K$ and the extractor algorithm of $\mathsf{NIZK}_{R_{\mathsf{dleq}}}$ i.e., $\mathsf{Ext}(R_{\mathsf{dleq}}, \pi_{dleq}, (G, K, \mathbf{G}, \mathsf{ring}, \mathsf{compk}, W, H_{\mathbf{G}}(m))) \to (\hat{x}, \widehat{\mathsf{oppk}})$ such that $\mathsf{compk} = \hat{x}G + \widehat{\mathsf{oppk}}\, K$ and $W = \hat{x}H_{\mathbf{G}}(m)$.

If X is an honest public key and $X = \hat{x}G$, $\mathcal{B}$ solves the CDH problem as follows: $W = \hat{x}hU$ where $h =$ `oracle_queries_gg`$[m]$. Since $X = rV$, $W = \hat{x}huG = rhuV$. 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 2.

If $\mathsf{counter}[m, \mathsf{ring}] = t \geq |\mathsf{ring}_{mal}|$, $\mathcal{B}$ obtains all the signatures $\{\sigma_i\}_{i=1}^t$ that make $\mathcal{B}$ increment $\mathsf{counter}[m, \mathsf{ring}]$ and solves the CDH problem as follows: Remark that this case happens when Sim aborts because of 1.

For all $\sigma_j = (\pi_{com_j}, \pi_{\mathsf{pk}_j}, \mathsf{compk}_j, W_j) \in \{\sigma_i\}_{i=1}^t$, $\mathcal{B}$ runs $\mathsf{Ext}(\mathcal{R}_{\mathsf{ring}}, \pi_{\mathsf{pk}_j}, (G, K, \mathbf{G}, \mathsf{ring}, \mathsf{compk}_j)) \to X_j, \mathsf{oppk}_j$ and adds X_j to a list $\mathcal{X}$ where $X_j \in \mathsf{ring}$ and $\mathsf{compk}_j = X_j + \mathsf{oppk}_j\, K$. One of the following cases happens:

- All X_j in $\mathcal{X}$ are different: If $\mathcal{B}$ is in this case, it means that there exists one public key $X_a \in \mathcal{X}$ which is honest. Then $\mathcal{B}$ runs the extractor algorithm of $\mathsf{NIZK}_{R_{\mathsf{dleq}}}$ i.e., $\mathsf{Ext}(R_{\mathsf{dleq}}, \pi_{com_a}, (G, K, \mathbf{G}, \mathsf{ring}, \mathsf{compk}_a, W_a, H_{\mathbf{G}}(m))) \to (\hat{x}_a, \widehat{\mathsf{oppk}}_a)$ such that $\mathsf{compk}_a = \hat{x}_aG + \widehat{\mathsf{oppk}}_a\, K$ and $W_a = \hat{x}_aH_{\mathbf{G}}(m)$. If $\mathcal{B}$ is in this case, $\hat{x}_aG \neq X_a$ because otherwise it would solve the CDH as described before. Therefore, $\mathsf{oppk}_a \neq \widehat{\mathsf{oppk}}_a$. Since $X_a + \mathsf{oppk}_a\, K = \hat{x}_aG + \widehat{\mathsf{oppk}}_a\, K$ and $X_a = r_aV$ where r_a is generated by $\mathcal{B}$ during the key generation process, $\mathcal{B}$ obtains a representation of $V = \gamma G + \delta K$ where $\gamma = \hat{x}_ar_a^{-1}$ and $\delta = (\widehat{\mathsf{oppk}}_a - \mathsf{oppk})\, r_a^{-1}$. Then $\mathcal{B}$ stores (γ, δ) to a list rep. If rep does not include another element $(\gamma', \delta') \neq (\gamma, \delta)$, $\mathcal{B}$ rewinds $\mathcal{A}$ to the beginning with a new random coin. Otherwise, it obtains (γ', δ') which is another representation of V i.e., $V = \gamma'G + \delta'K$. Thus, $\mathcal{B}$ can find discrete logarithm of V on base G which is $v = \gamma + \delta\theta$ where $\theta = (\gamma - \gamma')(\delta' - \delta)^{-1}$. $\mathcal{B}$ outputs vU as a CDH solution and the simulation ends.
- There exists at least two $X_a, X_b \in \mathcal{X}$ where $X_a = X_b$. $\mathcal{B}$ runs the extractor algorithm of $\mathsf{NIZK}_{R_{\mathsf{dleq}}}$ i.e., $\mathsf{Ext}(R_{\mathsf{dleq}}, \pi_{com_a}, (G, K, \mathbf{G}, \mathsf{ring}, \mathsf{compk}_a, W_a, H_{\mathbf{G}}(m))) \to (\hat{x}_a, \widehat{\mathsf{oppk}}_a)$ and $\mathsf{Ext}(R_{\mathsf{dleq}}, \pi_{com_b}, (G, K, \mathbf{G}, \mathsf{ring}, \mathsf{compk}_b, W_b, H_{\mathbf{G}}(m))) \to (\hat{x}_b, \widehat{\mathsf{oppk}}_b)$ such that $\mathsf{compk}_a = \hat{x}_aG + \widehat{\mathsf{oppk}}_a\, K \mathsf{compk}_b = \hat{x}_bG + \widehat{\mathsf{oppk}}_b\, K$ and $W_a = \hat{x}_aH_{\mathbf{G}}(m), W_b = \hat{x}_bH_{\mathbf{G}}(m)$. Since $W_a \neq W_b$, $\hat{x}_a \neq \hat{x}_b$. Therefore, $\mathcal{B}$ can obtain two different and non trivial representation of $X_a = X_b$ i.e., $X_a = X_b = \hat{x}_aG + (\widehat{\mathsf{oppk}}_a - \mathsf{oppk}_a)\, K = \hat{x}_bG + (\widehat{\mathsf{oppk}}_b - \mathsf{oppk}_b)\, K$. Thus, $\mathcal{B}$ finds the discrete logarithm of $K = U$ in base G which is $u = \frac{\hat{x}_a - \hat{x}_b}{\widehat{\mathsf{oppk}}_a - \mathsf{oppk}_a - \widehat{\mathsf{oppk}}_b + \mathsf{oppk}_b}$. $\mathcal{B}$ 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 except with a negligible probability.

This completes the security proof of our ring VRF protocol.

B Ring VRF Variations

In this section, we give ring VRF functionalities which give more security properties than the basic ring VRF functionality $\mathcal{F}_{\mathsf{rvrf}}$ that we define in Figure 1.

B.1 Ring VRF with Associated DATA

We define a variation of ring VRF which signs also an associated data (aux) along with a message. It is very similar to $\mathcal{F}_{\mathsf{rvrf}}$. It additionally requires unforgeability notion for ass as well. See Figure 7 for the details.

$\mathcal{F}_{\mathsf{rvrf}}^{\mathsf{aux}}$ runs two PPT algorithms Gen_{sign} during the execution.

Key Generation. Same as in $\mathcal{F}_{\mathsf{rvrf}}$.
Malicious Ring VRF Evaluation. Same as in $\mathcal{F}_{\mathsf{rvrf}}$.
Honest Ring VRF Signature. upon receiving a message $(\mathsf{sign}, \mathsf{sid}, \mathsf{ring}, \mathsf{pk}_i, m, \underline{aux})$ 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 the table $\texttt{verification\_keys}$. If that wasn't the case, just ignore the request. If there exists no W' such that $\texttt{anonymous\_key\_map}[m, W'] = \mathsf{pk}_i$, let $W \leftarrow \mathcal{S}_W$. Then, let $y \leftarrow\$ \mathcal{S}_{eval}$ and set $\texttt{anonymous\_key\_map}[W] = (m, \mathsf{pk}_i)$ and set $\texttt{evaluations}[m, W] = y$. Obtain W, y where $\texttt{evaluations}[m, W] = y$, $\texttt{anonymous\_key\_map}[m, W] = \mathsf{pk}_i$ and run $\mathsf{Gen}_{sign}(\mathsf{ring}, W, m, \underline{aux}) \to \sigma$. Verify that $[m, \underline{aux}, W, \sigma, 0]$ is not recorded. If not, abort. Otherwise, record $[m, \underline{aux}, W, \sigma, 1]$. Return $(\mathsf{signature}, \mathsf{sid}, \mathsf{ring}, W, m, \underline{aux}, y, \sigma)$ to P_i.
Ring VRF Verification. Same as in $\mathcal{F}_{\mathsf{rvrf}}$ except that $\mathcal{F}_{\mathsf{rvrf}}^{\mathsf{aux}}$ checks records $[m, \underline{aux}, W, \mathsf{ring}, \sigma, b]$ in the places where $\mathcal{F}_{\mathsf{rvrf}}$ checks $[m, W, \mathsf{ring}, \sigma, b]$.

Fig. 7. Functionality $\mathcal{F}_{\mathsf{rvrf}}^{\mathsf{aux}}$.

B.2 Secret Ring VRF

We also define another version of $\mathcal{F}_{\mathsf{rvrf}}$ that we call $\mathcal{F}_{\mathsf{rvrf}}^{s}$. $\mathcal{F}_{\mathsf{rvrf}}^{s}$ 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 $\mathcal{F}_{\mathsf{rvrf}}^{s}$ generates a proof. Proving works as $\mathcal{F}_{zk}$ [20] except that a part of the witness (η) is generated randomly by the functionality. $\mathcal{F}_{\mathsf{rvrf}}^{s}$ is useful in applications where a party wants to show that the random output y satisfies a certain relation without revealing his identity.

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

Secret Element Generation of Malicious Parties. upon receiving a message $(\mathsf{secret\_rand}, \mathsf{sid}, \mathsf{ring}, \mathsf{pk}, W, m)$ from Sim, verify that $\texttt{anonymous\_key\_map}[m, W] = \mathsf{pk}_i$. If that was not the case, just ignore the request. If $\texttt{secrets}[m, W]$ is not defined, obtain $y = \texttt{evaluations}[m, W]$. Then, run $\mathsf{Gen}_\eta(m, \mathsf{pk}_i, y) \to \eta$ and store $\texttt{secrets}[m, W] = \eta$. Obtain $\eta = \texttt{secrets}[m, W]$ 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 $\texttt{anonymous\_key\_map}[m, W] = \mathsf{pk}_i$. If that was not the case, just ignore the request. If $\texttt{secrets}[m, W]$ is not defined, run $\mathsf{Gen}_\eta(m, \mathsf{pk}_i, y) \to \eta$ and store $\texttt{secrets}[m, W] = \eta$. Obtain $\eta \leftarrow \texttt{secrets}[m, W]$ and $y \leftarrow \texttt{evaluations}[m, W]$. If $((m, y), (\eta, \mathsf{pk}_i)) \in \mathcal{R}$, run $\mathsf{Gen}_\pi(W, m) \to \pi$ and add π to a list $\texttt{zkproofs}[m, W]$. 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 $\pi \in \texttt{zkproofs}[m, W, \mathsf{ring}]$, set $b = 1$.
- else if $\texttt{secrets}[W, m] = \eta$ and $((m, y, \mathsf{ring}), (\eta, \mathsf{pk}_i)) \in \mathcal{R}$, set $b = 1$ and add to the list $\texttt{zkproofs}[m, W, \mathsf{ring}]$.
- else set $b = 0$.

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

Fig. 8. Functionality $\mathcal{F}^s_{\mathsf{rvrf}}$.