Ring Verifiable Random Function

No Author Given

No Institute Given

Abstract. Anonymized ring VRFs are ring signatures that prove correct evaluation of some authorized signer's PRF while hiding the specific signer's identity within some set of possible signers, known as the ring. We propose ring VRFs as a natural fulcrum around which a diverse array of zkSNARK circuits turn, making them an ideal target for optimization and eventually standards.

We show how rerandomizable Groth16 zkSNARKs transform into reusable zero-knowledge continuations, and build a ring VRF that amortizes expensive ring membership proofs across many ring VRF signatures. In fact, our ring VRF needs only eight $\mathcal{E}_1$ and two $\mathcal{E}_2$ scalar multiplications, making it the only ring signature with performance competitive with constructions like group signatures.

Ring VRFs produce a unique identity for any give context but which remain unlinkable between different contexts. These unlinkable but unique pseudonyms provide a far better balance between user privacy and service provider or social interests than attribute based credentials like IRMA.

Ring VRFs support anonymously rationing or rate limiting resource consumption that winds up vastly more efficient than purchases via monetary transactions.

1 Introduction

Anonymous credentials is the study of weakenings of anonymity, both the cryptographic techniques and the social implications. We answer four questions:

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

Zero-knowledge continuations: Rerandomizable zkSNARKs like Groth16 [6] admit a transformation of a valid proof into another valid but unlinkable proof of the exact same statement. In practice, rerandomization was never deployed because the public inputs link the usages.

We demonstrate in §6 a simple transformation of any Groth16 zkSNARK into a *zero-knowledge continuation* whose public inputs become 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.

As recursive SNARKs shall remain extremely slow, we expect zero-knowledge continuations via rerandomization become essential for zkSNARKs used outside the crypto-currency space.

Ring VRFs: A *ring verifiable random function* (ring VRF) is a ring signature that proves correct evaluation of some pseudo-random function (PRF) determined by the actual key pair used in signing (see §3). 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.

As the PRF output is uniquely determined by the signed message and signers 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.

Identity uses: As an identity system, ring VRFs evaluated on a specific context or domain name output a unique identity for the user at that domain or context (see §**??**), which thereby prevents Sybil behavior and permits banning specific users. Yet users' activities remain unlinkable across distinct contexts or domains, which supports diverse ethical identity usages.

We contrast this ethically straightforward ring VRF based identity with the ethically problematic case of attribute based credential schemes like IRMA [3], which are now marketed as an online privacy solution. IRMA could improve privacy in narrow situations of course, but overall attribute based credentials should *never* be considered fit for general purpose usage, like the prevention of Sybil behavior.

Aside from general purpose identity, our existing offline verification processes often better protect user 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 [9], but broadly speaking they all bring matching harmful uses. As an example, if users could easily prove their employment online when applying for a bank account, then job application sites could similarly demand proof of current employment, a clear injustice.

In general, abuse risks dictate that IRMA verifiers should be tightly controlled by legislation, which becomes difficult 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 [5,2] 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 like IRMA [3] or group signatures [8]. We focus exclusively upon ring VRFs for brevity, and because alone ring VRFs contextual linkability covers more important use cases.

Rationing uses: Ring VRFs yield rate limiting or rationing systems, which work similarly to identity applications, except the VRF input should also include a approximate date and a bounded counter, and then their outputs should be tracked as nullifiers. Yet, these nullifiers need only temporarily storage, which improves efficiency over anonymous money schemes like ZCash and blind signed tokens.

We must 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 certificates concerns because they permit direct public audits of ring membership.

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 then help avoid ration card fraud, and thereby reduce social unrest, while also protecting essential privacy.

Ring VRFs need heavier verifiers than single-use token credentials based on OPRFs [4] or blind signatures. Yet, ring VRFs avoid the schemes separate issuance phase entirely, which reduces complexity, simplifies scaling, and increases flexibility.

In particular, if governments issue ration cards based upon ring VRFs then these credentials could safely support free tiers in online services and games, and other limited advertiser promotions, as well as identity applications like simple logins and 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 §**??** 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.

2 Preliminaries

2.1 Non-Interactive Zero-Knowledge Proofs

Definition 1. *Let $\mathfrak{R}$ be a relation generator given security parameter λ which returns polynomial time decidable binary relation $\mathcal{R}$. Given a relation $\mathcal{R}$ generated by $\mathfrak{R}$, a non-interactive proving system consists of the following algorithms:*

- $\mathsf{NIZK.Prove}(\mathcal{R}, (\omega, x)) \to \pi$*: Given $\mathcal{R}$ a witness and statement pair $(\omega, x) \in \mathcal{R}$, it outputs a proof π.*
- $\mathsf{NIZK.Verify}(\mathcal{R}, x, \pi) \to 0/1$*: Given $\mathcal{R}$, a statement x and a proof π, it outputs either 0 meaning not verified or 1 meaning verified.*

Definition 2. *A non-interactive proving system is a NIZK if the following properties are satisfied:*

- ***Completeness:*** *A non-interactive proving system is complete if given any statement $x \in \mathcal{L}$ where $\mathcal{L} = \{x : (\omega, x) \in \mathcal{R}\}$, for all relations $\mathcal{R}$ generated by $\mathfrak{R}(\lambda)$, $(\omega, x) \in \mathcal{R}$,*

$$\Pr[\mathsf{NIZK.Prove}(\mathcal{R}, (\omega, x)) \to \pi : \mathsf{NIZK.Verify}(\mathcal{R}, x, \pi) \to 1] = 1.$$

- ***Zero-knowledge:*** *A non-interactive proving system is zero-knowledge if for all relations $\mathcal{R}$ generated by $\mathfrak{R}(\lambda)$, $(\omega, x) \in \mathcal{R}$ and for all polynomial adversaries V^*, there exists a simulator algorithm* $\mathsf{NIZK.Simulate}$ *such that*

$$\begin{aligned}&\Pr[\mathsf{NIZK.Simulate}(\mathcal{R}, x) \to \pi : V^*(\mathcal{R}, x, \pi) \to 1] \\ -&\Pr[\mathsf{NIZK.Prove}(\mathcal{R}, (\omega, x)) \to \pi : V^*(\mathcal{R}, x, \pi) \to 1] = \mathsf{negl}(\lambda)\end{aligned}$$

- ***Knowledge Soundness:*** *A non-interactive proving system is knowledge sound if for all relations $\mathcal{R}$ generated by $\mathfrak{R}(\lambda)$, $(\omega, x) \in \mathcal{R}$ and for all non-uniform polynomial adversaries $P^*(\mathcal{R})$ which outputs π for a statement x verified by* $\mathsf{NIZK.Verify}$*, there exists a non-uniform polynomial time extractor algorithm* Ext *such that*

$$\Pr[\mathcal{R} \leftarrow \mathfrak{R}(\lambda), ((x, \pi); \omega) \leftarrow (P^* || \mathsf{Ext})(\mathcal{R}) : (\omega, x) \notin \mathcal{R}, \mathsf{NIZK.Verify}(\mathcal{R}, x, \pi) \to 1] = \mathsf{negl}(\lambda)$$

where $((x, \pi); \omega) \leftarrow (P^* || \mathsf{Ext})(\mathcal{R})$ shows the joint execution on the same input $\mathcal{R}$ where P^*'s output is (x, π) and Ext*'s output is ω.*

2.2 Universal Composability

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. $\mathcal{Z}$ initiates ITMs and $\mathcal{A}$ with arbitrary inputs and then terminates them to collect the outputs. An ITM that is initiated by $\mathcal{Z}$ is called ITM instance (ITM). 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:

ϕ in the 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 $\{\mathsf{EXEC}(\lambda, z)_{\phi,\mathcal{A},\mathcal{Z}}\}_{z\in\{0,1\}^*}$.

ϕ in the ideal world: $\mathcal{Z}$ initiates ITMs and a simulator Sim to contact with the ideal functionalityt $\mathcal{F}$ 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 3 (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 if ϕ UC-realizes $\mathcal{F}$ i.e., if for all PPT adversaries $\mathcal{A}$, there exists a simulator* Sim *such that for any environment $\mathcal{Z}$,*

$$\mathsf{EXEC}_{\phi,\mathcal{A},\mathcal{Z}} \approx \mathsf{EXEC}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}$$

where $\approx$ means indistinguishable.

Definition 4 (UC-Security of ϕ in the hybrid world). *Given a real world protocol ϕ which runs some (polynomially many) functionalities $\{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_k\}$ in the ideal world and an ideal functionality $\mathcal{F}$ for the protocol ϕ, we call that ϕ is UC-secure in the hybrid model $\{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_k\}$ if ϕ UC-realizes $\mathcal{F}$ if for all PPT adversaries $\mathcal{A}$, there exists a simulator* Sim *such that for any environment $\mathcal{Z}$,*

$$\mathsf{EXEC}_{\phi,\mathcal{A},\mathcal{Z}} \approx \mathsf{EXEC}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}$$

3 Ring Verifiable Random Function

In this section, we describe our new cryptographic primitive ring VRF. It acts a ring signature [] but additionally it lets a party generate a random element. In this sense, a ring VRF operates like a VRF but only proves its key comes from a specific list without giving any information about which specific key.

Definition 5 (Ring-VRF (rVRF)). *A ring VRF is a VRF with a function $F(.) : \{0,1\}^* \to \{0,1\}^{\ell_{\mathsf{rVRF}}}$ and with the following algorithms:*

- $\mathsf{rVRF.KeyGen}(1^\kappa) \to (\mathsf{sk}, \mathsf{pk})$ *where κ is the security parameter,*

Given list of public keys $\mathcal{PK} = \{\mathsf{pk}_1, \mathsf{pk}_2, \ldots, \mathsf{pk}_n\}$, a message $m \in \{0,1\}^{\ell_m}$

- $\mathsf{rVRF.Sign}(\mathsf{sk}_i, \mathcal{PK}, m) \to (\sigma, W)$ *where σ is a signature of the message m signed by $\mathsf{sk}_i, \mathcal{PK}$ and W is an anonymous key.*
- $\mathsf{rVRF.Verify}(\mathsf{comring}, W, m, \sigma) \to (b, y)$ *where $b \in \{0,1\}$ and $y \in \{0,1\}^{\ell_{\mathsf{rVRF}}} \cup \{\perp\}$. $b = 1$ means verified and $b = 0$ means not verified.*
- $\mathsf{rVRF.Link}(\mathsf{sk}_i, \mathcal{PK}, W, m, \sigma) \to \hat{\sigma}$ *where $\hat{\sigma}$ is a signature that links signer of the ring signature σ.*
- $\mathsf{rVRF.LinkVerify}(\mathsf{pk}_i, \mathcal{PK}, W, m, \sigma, \hat{\sigma}) \to b$ *where $b \in \{0,1\}$. $b = 1$ means verified and $b = 0$ means not verified.*

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

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 $\mathtt{verification\_keys}$, under P_i, the value pk. Return $(\mathsf{verificationkey}, \mathsf{sid}, \mathsf{pk})$ to P_i.

Malicious Key Generation. upon receiving a message $(\mathsf{keygen}, \mathsf{sid}, \mathsf{pk})$ from Sim, verify that pk was not yet recorded, and if so record in the table $\mathtt{verification\_keys}$ the value pk under Sim. Else, ignore the message.

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

Malicious Ring VRF Evaluation. upon receiving a message $(\mathsf{eval}, \mathsf{sid}, \mathcal{PK}, \mathsf{pk}_i, W, m)$ from Sim, verify that pk_i has not been recorded under an honest party's identity. If it is the case record in the table $\mathtt{verification\_keys}$ the value pk_i under Sim if it is not in $\mathtt{verification\_keys}$. Else, ignore the request. If $\mathsf{counter}[m, \mathcal{PK}]$ does not exist, initiate $\mathsf{counter}[m, \mathcal{PK}] = 0$. If there exists $W' \neq W$ where $\mathtt{anonymous\_key\_map}[m, W, \mathcal{PK}] = \mathtt{anonymous\_key\_map}[m, W', \mathcal{PK}]$, then abort. Otherwise, check if $\mathtt{Out}[m, W, \mathcal{PK}]$ is defined. If it is not defined, let $y \leftarrow\!\$\, \{0,1\}^{\ell_{\mathsf{rVRF}}}$. Then, set $\mathtt{Out}[m, W, \mathcal{PK}] = y$, $\mathtt{anonymous\_key\_map}[m, W, \mathcal{PK}] = \mathsf{pk}_i$ Return $(\mathsf{evaluated}, \mathsf{sid}, \mathcal{PK}, m, W, \mathtt{Out}[m, W, \mathcal{PK}])$ to P_i.

Honest Ring VRF Signature and Evaluation. upon receiving a message $(\mathsf{sign}, \mathsf{sid}, \mathcal{PK}, \mathsf{pk}_i, m)$ from P_i, verify that $\mathsf{pk}_i \in \mathcal{PK}$ and that there exists a public key pk_i associated to P_i in the table $\mathtt{verification\_keys}$. If that wasn't the case, just ignore the request. If there exists no W' such that $\mathtt{anonymous\_key\_map}[m, \mathcal{PK}, W'] = \mathsf{pk}_i$, run $\mathsf{Gen}_W(\mathcal{PK}, \mathsf{pk}_i, m) \rightarrow W$. Then, let $y \leftarrow\!\$\, \{0,1\}^{\ell_{\mathsf{rVRF}}}$ and set $\mathtt{anonymous\_key\_map}[m, W, \mathcal{PK}] = \mathsf{pk}_i$ and set $\mathtt{Out}[m, W, \mathcal{PK}] = y$. Obtain W, y where $\mathtt{Out}[m, W, \mathcal{PK}] = y$, $\mathtt{anonymous\_key\_map}[m, W, \mathcal{PK}] = \mathsf{pk}_i$ and run $\mathsf{Gen}_{sign}(\mathcal{PK}, W, m) \rightarrow \sigma$. Verify that $[m, W, \mathcal{PK}, \sigma, 0]$ is not recorded. If it is recorded, abort. Otherwise, record $[m, W, \mathcal{PK}, \sigma, 1]$. Return $(\mathsf{signature}, \mathsf{sid}, \mathcal{PK}, W, m, y, \sigma)$ to P_i.

Ring VRF Verification. upon receiving a message $(\mathsf{verify}, \mathsf{sid}, \mathcal{PK}, W, m, \sigma)$ from a party, relay the message $(\mathsf{verify}, \mathsf{sid}, \mathcal{PK}, W, m, \sigma)$ to Sim and receive back the message $(\mathsf{verified}, \mathsf{sid}, \mathcal{PK}, W, m, \sigma, b_{\mathsf{Sim}}, \mathsf{pk}_{\mathsf{Sim}})$. Then do the following:

Cond.- 1 If there exits a record $[m, W, \mathcal{PK}, \sigma, b']$, set $b = b'$. (This condition guarantees the completeness and consistency.)

Cond.- 2 Else if $\mathtt{anonymous\_key\_map}[m, W, \mathcal{PK}]$ is an honest verification key where there exists a record $[m, W, \mathcal{PK}, \sigma', 1]$ for any σ', then let $b = b_{\mathsf{Sim}}$ and record $[m, W, \sigma, b_{\mathsf{Sim}}]$. (This condition guarantees that if m is signed by an honest party for the ring $\mathcal{PK}$ at some point and the signature is $\sigma' \neq \sigma$, then the decision of verification is up to the adversary)

Cond.- 3 Else if $\mathsf{counter}[m, \mathcal{PK}] > |\mathcal{PK}_{mal}|$ where $\mathcal{PK}_{mal}$ is a set of malicious keys in $\mathcal{PK}$, set $b = 0$ and record $[m, \mathcal{PK}, W, \mathcal{PK}, \sigma, 0]$. (This condition guarantees uniqueness meaning that the number of verifying outputs that Sim can generate for $m, \mathcal{PK}$ is at most the number of malicious keys in $\mathcal{PK}$.).

Cond.- 4 Else if $\mathsf{pk}_{\mathsf{Sim}}$ is an honest verification key, set $b = 0$ and record $[m, \mathcal{PK}, W, \mathcal{PK}, \sigma, 0]$. (This condition guarantees unforgeability meaning that if an honest party never signs a message m for a ring $\mathcal{PK}$)

Cond.- 5 Else set $b = b_{\mathsf{Sim}}$.

In the end, if $b = 0$, let $\mathsf{Out} = \perp$. Otherwise, it does the following:

- if $\mathtt{Out}[m, W, \mathcal{PK}]$ is not defined, set $\mathtt{Out}[m, W, \mathcal{PK}] \leftarrow\!\$\, \{0,1\}^{\ell_{\mathsf{rVRF}}}$, $\mathtt{anonymous\_key\_map}[m, W, \mathcal{PK}] = \mathsf{pk}_{\mathsf{Sim}}$. If $\mathsf{counter}[m, \mathcal{PK}]$ is not defined, set $\mathsf{counter}[m, \mathcal{PK}] = 0$. Then increment $\mathsf{counter}[m, \mathcal{PK}, \mathsf{pk}_{\mathsf{Sim}}]$. Set $\mathsf{Out} = \mathtt{Out}[m, W, \mathcal{PK}]$.
- otherwise, set $\mathsf{Out} = \mathtt{Out}[m, W, \mathcal{PK}]$.

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

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

This part of $\mathcal{F}_{\mathsf{rvrf}}$ for the parties who want to show that they generate a particular ring signature.

Linking signature. upon receiving a message $(\mathsf{link}, \mathsf{sid}, \mathcal{PK}, \mathsf{pk}_i, W, m, \sigma)$ from P_i, check that pk_i is associated to P_i in `verification_keys`, $\mathsf{pk}_i \in \mathcal{PK}$, `anonymous_key_map`$[m, W, \mathcal{PK}] = \mathsf{pk}_i$ and check whether $[m, W, \mathcal{PK}, \sigma, 1]$ is stored. If any of them fails, ignore the request. Otherwise, send $(\mathsf{link}, \mathsf{sid}, \mathcal{PK}, W, m, y)$ to Sim. Upon receiving $(\mathsf{linkproof}, \mathsf{sid}, \mathcal{PK}, W, m, y, \hat{\sigma})$ from Sim, verify that $[m, \mathcal{PK}, \mathsf{pk}_i, \sigma, \hat{\sigma}, 0]$ is not stored in `links`. If not, abort. Otherwise, record $\hat{\sigma}$ to $[m, \mathcal{PK}, \mathsf{pk}_i, \sigma, \hat{\sigma}, 1]$ to `links` and return $(\mathsf{linked}, \mathsf{sid}, \mathcal{PK}, \mathsf{pk}_i, W, m, y, \sigma, \hat{\sigma})$ to P_i.

Linking verification. upon receiving a message $(\mathsf{verifylink}, \mathsf{sid}, \mathsf{pk}_i, \mathcal{PK}, W, m, \sigma, \hat{\sigma})$ from any party forward the message to the simulator and receive back the message $(\mathsf{verified}, \mathsf{sid}, \mathsf{pk}_i, \mathcal{PK}, W, m, \sigma, \hat{\sigma}, b_{\mathsf{Sim}})$. Then do the following:

- If there exits a record $[m, \mathcal{PK}, \mathsf{pk}_i, \sigma, \hat{\sigma}, 1]$ in `links`, set $b = 1$. (This condition guarantees the completeness.)
- Else if pk_i is a key of an honest party and there exits no record such as $[m, \mathcal{PK}, \mathsf{pk}_i, \sigma, \hat{\sigma}', 1]$ for any $\hat{\sigma}'$, then set $b = 0$ and record $[m, \mathcal{PK}, \mathsf{pk}_i, \sigma, \hat{\sigma}, 0]$. (This condition guarantees unforgeability meaning that if an honest party never signs a message m in the linking signature, then the verification fails.)
- Else if there exists a record such as $[m, \mathcal{PK}, \mathsf{pk}_i, \sigma, \hat{\sigma}, b']$, set $b = b'$.
- Else set $b = b_{\mathsf{Sim}}$ and record $[m, \mathcal{PK}, \mathsf{pk}_i, \sigma, \hat{\sigma}, 1]$.

Return $(\mathsf{verified}, \mathsf{sid}, \mathsf{pk}_i, \mathcal{PK}, m, \hat{\sigma}, b)$. to the party.

Fig. 2. Functionality $\mathcal{F}_{\mathsf{rvrf}}$.

3.1 Ring VRF in the UC Model

In this section, we describe the security of our new cryptographic primitive ring VRF in the UC model.

Ring VRF Functionality: We define the ring VRF functionality $\mathcal{F}_{\mathsf{rvrf}}$ in Figure 2. The functionality lets parties generate a key (Key Generation), evaluate a message with the party's key (Ring VRF Evaluation), sign a message by one of the keys (Ring VRF signature) and verify the signature and obtain the evaluation output without knowing the key used for the signature and evaluation (Ring VRF Verification).

We also define linking procedures in $\mathcal{F}_{\mathsf{rvrf}}$ to link a signature with its associated key. So, if a party wants to reveal its identity at some point, it can use the linking process to show that the evaluation is executed with its key (Linking Signature). Later on, anyone can verify the linking signature (Linking Verification).

In a nutshell, the functionality $\mathcal{F}_{\mathsf{rvrf}}$, when given as input a message m and a key set $\mathcal{PK}$ (that we call a ring) of party, allows to create n possible different outputs pseudo-random that appear independent from the inputs. The output can be verified to have been computed correctly by one of the participants in $\mathcal{PK}$ without revealing who they are. At a later stage, the author of the ring VRF output can prove that the output was generated by him and no other participant could have done so.

There are four properties that we want to achieve for a ring VRF protocol in the real world. In short, these properties are as follows: The first property is pseudorandomness meaning that the output of message m and public key pk is pseudorandom. The second property is anonymity meaning a signature σ verified by a ring $\mathcal{PK}$ does not give any information about its signer. The third property is unforgeability meaning that if a party with a public key pk never signs a message m for a ring $\mathcal{PK}$, then no party is able to generate a signature of m for $\mathcal{PK}$ signed by pk. The last property is uniqueness meaning that the number of verified outputs via signatures for a message and ring $\mathcal{PK}$ is not more than $|\mathcal{PK}|$.

R- 1 In classical VRF, a VRF F is a deterministic function which maps a message and a public key to a random output. While in ring VRF, a message, a public key and a ring map to a random value, the verification algorithm of a ring VRF does not take the key as an input because it should be hidden. Therefore, the verification should be executed without the public key.

So, the functionality $\mathcal{F}_{\mathsf{rvrf}}$ needs to find a way to verify the ring VRF output of a message, a public key and a ring map without knowing the public key. Because of this, $\mathcal{F}_{\mathsf{rvrf}}$ generates an anonymized key W for each evaluation so that a message m and W maps to the random output. One can imagine this as if a VRF output is generated with the input message m and the key W as in classical VRF i.e., $F(m, W)$.

R- 2 If an honest party signs a message for a ring and obtains a signature, $\mathcal{F}_{\mathsf{rvrf}}$ allows the simulator to generate another signature in Cond.- 2 if the simulator wants. We remark that this is not a security issue because an honest party has already committed to sign the message. A similar condition exists in the EUF-CMA secure signature functionality $\mathcal{F}_{\mathsf{sig}}$ [**?**].

R- 3 Cond.- 5 of the ring VRF verification process covers the case where the adversary decides whether accepting the signature generated for its key if it could be a valid signature for the ring i.e., the malicious key is in the ring and the anonymous key in the verification request is unique.

R- 4 The linking signature and the linking verification works similar to the EUF-CMA secure signature functionality $\mathcal{F}_{\mathsf{sig}}$ [**?**].

Definition 6 (Anonymous $\mathcal{F}_{\mathsf{rvrf}}$). *We call that $\mathcal{F}_{\mathsf{rvrf}}$ is anonymous if the outputs of Gen_{sign} and Gen_W are pseudo-random.*

Below, we define the real-world execution of a ring VRF.

Definition 7 (Ring-VRF (rVRF)). *A ring VRF is a VRF with a function $F(.) : \{0,1\}^* \to \{0,1\}^{\ell_{\mathsf{rVRF}}}$ and with the following algorithms:*

- $\mathsf{rVRF.KeyGen}(1^\kappa) \to (\mathsf{sk}, \mathsf{pk})$ *where κ is the security parameter,*

Given list of public keys $\mathcal{PK} = \{\mathsf{pk}_1, \mathsf{pk}_2, \ldots, \mathsf{pk}_n\}$, a message $m \in \{0,1\}^{\ell_m}$

- $\mathsf{rVRF.Sign}(\mathsf{sk}_i, \mathcal{PK}, m) \to (\sigma, W)$ *where σ is a signature of the message m signed by $\mathsf{sk}_i, \mathcal{PK}$ and W is an anonymous key.*
- $\mathsf{rVRF.Verify}(\mathcal{PK}, W, m, \sigma) \to (b, y)$ *where $b \in \{0,1\}$ and $y \in \{0,1\}^{\ell_{\mathsf{rVRF}}} \cup \{\perp\}$. $b = 1$ means verified and $b = 0$ means not verified.*
- $\mathsf{rVRF.Link}(\mathsf{sk}_i, \mathcal{PK}, W, m, \sigma) \to \hat{\sigma}$ *where $\hat{\sigma}$ is a signature that links signer of the ring signature σ.*
- $\mathsf{rVRF.LinkVerify}(\mathsf{pk}_i, \mathcal{PK}, W, m, \sigma, \hat{\sigma}) \to b$ *where $b \in \{0,1\}$. $b = 1$ means verified and $b = 0$ means not verified.*

4 Our Ring VRF Construction rVRF

We instantiate parameter generation by constructing a group $\mathbb{G}$ of order p and two generators $G_1, G_2 \in \mathbb{G}$. We consider three hash functions: $\mathsf{H}, \mathsf{H}', \mathsf{H}_{\mathsf{com}} : \{0,1\}^* \to \mathbb{F}_p$ and a hash-to-group function $\mathsf{H}_{\mathbb{G}} : \{0,1\}^* \to \mathbb{G}$ 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 $X = xG_1$. In the end, it outputs $\mathsf{sk} = x$ and $\mathsf{pk} = X$.
- $\mathsf{rVRF.Sign}(\mathsf{sk}, \mathcal{PK}, m)$: It lets $P = \mathsf{H}_{\mathbb{G}}(m, \mathcal{PK})$ and computes the anonymous key $W = xP$. The signing algorithm works as follows:
  - It first commits to its secret key x i.e., $C = X + \beta G_2$ where $\beta \leftarrow\!\!\$\, \mathbb{F}_p$.

- It generates a Schnorr proof π_{com} showing the following relation, i.e., $\mathsf{NIZK.Prove}(\mathcal{R}_{\mathsf{com}}, ((x,\beta),(G_1,G_2,\mathbb{G},C,W,P))) \rightarrow \pi_{com}$

$$\mathcal{R}_{\mathsf{com}} = \{((x,\beta),(G_1,G_2,\mathbb{G},C,W,P)) : C = xG_1 + \beta G_2, W = xP\} \tag{1}$$

  Here Prove algorithm runs a non-interactive Schnorr proof with the Fiat-Shamir transform: Sample random $r_1, r_2 \leftarrow \mathbb{F}_p$. Let $R = r_1G_1 + r_2G_2$, $R_m = r_1P$, and $c = \mathsf{H}'(\mathcal{PK}, m, W, C, R, R_m)$. Set $\pi_{com} = (c, s, \delta)$ where $s = r_1 - cx$ and $\delta = r_2 - c\beta$.
- It generates the second proof π_{pk} for the following relation with the witness $(\mathcal{PK}, x, \beta)$.

$$\mathcal{R}_{\mathsf{pk}} = \{(X,\beta),(G_1,G_2,\mathbb{G},\mathcal{PK},C)) : C - \beta G_2 = X \in \mathcal{PK}\} \tag{2}$$

  The second proof π_{pk} is generated by running $\mathsf{NIZK.Prove}(\mathcal{R}_{\mathsf{pk}}, ((X,\beta),(G_1,G_2,\mathbb{G},\mathcal{PK},C)))$

In the end, $\mathsf{rVRF.Sign}$ outputs σ, W where $\sigma = (\pi_{com}, \pi_{pk}, C)$.

– $\mathsf{rVRF.Verify}(\mathcal{PK}, W, m, \sigma)$: Given $\sigma = (\pi_{com}, \pi_{pk}, C)$ and $\mathcal{PK}, W$, it runs $\mathsf{NIZK.Verify}(\mathcal{R}_{\mathsf{com}}, (G_1,G_2,\mathbb{G},C,W,P), \pi_{com})$ where $P = \mathsf{H}_{\mathbb{G}}(m, \mathcal{PK})$. $\mathsf{NIZK.Verify}$ for $\mathcal{R}_{\mathsf{com}}$ works as follows: $\pi_{com} = (c, s, \delta)$, it lets $R' = sG_1 + \delta G_2 + cC$ and $R'_m = s\mathsf{H}_{\mathbb{G}}(m, \mathcal{PK}) + cW$. It returns true if $c = \mathsf{H}'(\mathcal{PK}, m, W, C, R', R'_m)$. If $\mathsf{NIZK.Verify}(\mathcal{R}_{\mathsf{com}}, (G_1,G_2,\mathbb{G},C,W,P), \pi_{com})$ outputs 1, it runs $\mathsf{NIZK.Verify}(\mathcal{R}_{\mathsf{pk}}, (G_1,G_2,\mathbb{G},\mathsf{root},C), \pi_{MT})$ where root is the Merkle tree root of the public keys $\mathcal{PK}$. If all verification algorithms verify, it outputs 1 and the evaluation value $y = \mathsf{H}(m, \mathcal{PK}, W)$. Otherwise, it outputs $(0, \perp)$.
– $\mathsf{rVRF.Link}(\mathsf{sk}, \mathcal{PK}, W, m, \sigma)$: It generates proof for the relation $\mathcal{R}_{dleq}$ i.e., $\mathsf{NIZK.Prove}(\mathcal{R}_{dleq}, (G_1, \mathbb{G}, X, P, W)) \rightarrow \hat{\sigma}$ where $P = \mathsf{H}_{\mathbb{G}}(m, \mathcal{PK})$

$$\mathcal{R}_{dleq} = \{(x, (G_1, \mathbb{G}, X, P, W)) : X = xG_1, W = xP\}$$

  In the end, it outputs $\hat{\sigma}$
– $\mathsf{rVRF.LinkVerify}(\mathsf{pk}, \mathcal{PK}, W, m, \sigma, \hat{\sigma})$: It outputs $\mathsf{NIZK.Verify}(\mathcal{R}_{dleq}, (G_1, \mathbb{G}, X, P, W), \hat{\sigma})$ where $P = \mathsf{H}_{\mathbb{G}}(m, \mathcal{PK})$.

5 Security Proof of Our Protocol in UC

Before we start to analyse our protocol, we should define the algorithms Gen_{sign} and Gen_W for $\mathcal{F}_{\mathsf{rvrf}}$. $\mathcal{F}_{\mathsf{rvrf}}$ that rVRF realizes runs Algorithm 1 to generate anonymous keys and Algorithm 2 to generate signatures.

Algorithm 1 $\mathsf{Gen}_W(\mathcal{PK}, \mathsf{pk}, m)$

1: $W \leftarrow\$ \mathbb{G}$
2: **return** W

Algorithm 2 $\mathsf{Gen}_{sign}(\mathcal{PK}, W, \mathsf{pk}, m)$

1: $c, s, \delta \leftarrow\$ \mathbb{F}_p$
2: $\beta \leftarrow\$ \mathbb{F}_p$
3: $C = \mathsf{pk} + \beta G_2$
4: $\pi_{com} \leftarrow (c, s, \delta)$
5: $\pi_{MT} \leftarrow \mathsf{NIZK.Prove}(\mathcal{R}_{\mathsf{pk}}, ((X,\beta),(G_1,G_2,\mathbb{G},\mathcal{PK},C)))$
6: **return** $\sigma = (\pi_{com}, \pi_{MT}, C, W)$

Clearly, Gen_W and Gen_{sign} satisfy the anonymity defined in Definition 6 because W and σ are generated independent from the public key pk.

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 1. *Assuming that $\mathsf{H}_{\mathbb{G}}, \mathsf{H}, \mathsf{H}'$ are random oracles, the DDH problem is hard in the group structure $(\mathbb{G}, G_1, G_2, p)$ and NIZK algorithms are zero-knowledge and knowledge sound, rVRF UC-realizes $\mathcal{F}_{\mathsf{rvrf}}$ according to Definition 3.*

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 $\mathsf{honest\_verification\_keys}$ and $\mathtt{verification\_keys}$ 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 $\mathsf{honest\_verification\_keys}$ which is stored as a key of P_i and adds X to $\mathsf{malicious\_verification\_keys}$.
- [**Simulation of the random oracles:**] We describe how Sim simulates the random oracles $\mathsf{H}_{\mathbb{G}}, \mathsf{H}$ against the real world adversaries. Sim simulates H' as a usual random oracle.
  Sim simulates the random oracle $\mathsf{H}_{\mathbb{G}}$ as described in Figure 3. It selects a random element h from $\mathbb{F}_p$ for each new input and outputs hG_1 as an output of the random oracle $\mathsf{H}_{\mathbb{G}}$. Thus, Sim knows the discrete logarithm of each random oracle output of $\mathsf{H}_{\mathbb{G}}$.

Oracle $\mathsf{H}_{\mathbb{G}}$
Input: $m, \mathcal{PK}$
if $\mathtt{oracle\_queries\_gg}[m, \mathcal{PK}] = \perp$
  $h \leftarrow\!\!\$\, \mathbb{F}_p$
  $P \leftarrow hG_1$
  $\mathtt{oracle\_queries\_gg}[m, \mathcal{PK}] := h$
else:
  $h \leftarrow \mathtt{oracle\_queries\_gg}[m, \mathcal{PK}]$
  $P \leftarrow hG_1$
return P

Fig. 3. The random oracle $\mathsf{H}_{\mathbb{G}}$

The simulation of the random oracle H is less straightforward (See Figure 4). Whenever an input $(m, \mathcal{PK}, W)$ is given to H, it first obtains the discrete logarithm h of $\mathsf{H}_{\mathbb{G}}(m, \mathcal{PK})$ from the $\mathsf{H}_{\mathbb{G}}$'s database. It needs this information to obtain a public key which could be used when running $\mathsf{rVRF.Sign}$. Remark that if W is a pre-output generated as described in $\mathsf{rVRF.Sign}$, it should be equal to $x\mathsf{H}_{\mathbb{G}}(m, \mathcal{PK}) = xhG_1$ where xG_1 is a public key. Therefore, the random oracle H obtains first a key $X^* = h^{-1}W$ to check whether it is an honest key. If X^* is not an honest key generated by Sim, Sim considers the input $(m, \mathcal{PK}, W)$ as an attempt to learn the output of the ring signature of the message m, the ring $\mathcal{PK}$ and the pre-output W. Remark that if X^* is not an honest key, Sim has a right to ask the output of m with the public key X^* and the ring $\mathcal{PK}$. Therefore, Sim sends the message $(\mathsf{eval}, \mathsf{sid}, \mathcal{PK}, W, m)$ to $\mathcal{F}_{\mathsf{rvrf}}$ and receives the message $(\mathsf{evaluated}, \mathsf{sid}, \mathcal{PK}, W, m, y)$. After learning the evaluation output y, it sets y as the answer of the random oracle H for the input $(m, \mathcal{PK}, W)$. If X^* is registered as an honest party's key, Sim outputs a randomly selected element.
- [**Simulation of** verify] Upon receiving $(\mathsf{verify}, \mathsf{sid}, \mathcal{PK}, W, m, \sigma)$ from the functionality $\mathcal{F}_{\mathsf{rvrf}}$, Sim runs the two NIZK verification algorithms run for $\mathcal{R}_{\mathsf{com}}, \mathcal{R}_{\mathsf{pk}}$ with the input $\mathcal{PK}, m, \sigma, W$

Oracle H
Input: $m, \mathcal{PK}, W$
if $\texttt{oracle\_queries\_h}[m, \mathcal{PK}, W] \neq \perp$
  return $\texttt{oracle\_queries\_h}[m, \mathcal{PK}, W]$
$P \leftarrow \mathsf{H}_{\mathbb{G}}(m, \mathcal{PK})$
$h \leftarrow \texttt{oracle\_queries\_gg}[m, \mathcal{PK}]$
$X^* := h^{-1}W$ // candidate verification key
if $X^* \notin$ honest_verification_keys
  send $(\mathsf{eval}, \mathsf{sid}, \mathcal{PK}, W, X^*, m)$ **to** $\mathcal{F}_{\mathsf{rvrf}}$
  receive $(\mathsf{evaluated}, \mathsf{sid}, \mathcal{PK}, W, m, y)$ **from** $\mathcal{F}_{\mathsf{rvrf}}$
  $\texttt{oracle\_queries\_h}[m, \mathcal{PK}, W] := y$
else
  $y \leftarrow\!\!\$\ \mathbb{F}_p$
  $\texttt{oracle\_queries\_h}[m, \mathcal{PK}, W] := y$
return $\texttt{oracle\_queries\_h}[m, \mathcal{PK}, W]$

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 $h = \texttt{oracle\_queries\_gg}[m, \mathcal{PK}]$ and sends $(\mathsf{verified}, \mathsf{sid}, \mathcal{PK}, W, m, \sigma, b_{\mathsf{Sim}}, X)$ to $\mathcal{F}_{\mathsf{rvrf}}$ and receives back $(\mathsf{verified}, \mathsf{sid}, \mathcal{PK}, W, m, \sigma, y, b)$.
  - 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 Cond.- 1, Cond.- 3 or Cond.- 4. If $\mathcal{F}_{\mathsf{rvrf}}$ is in Cond.- 3, it means that $\mathsf{counter}[m, \mathcal{PK}] > |\mathcal{PK}_m|$. If $\mathcal{F}_{\mathsf{rvrf}}$ is in Cond.- 4, it means that X belongs to an honest party but this honest party never signs m for the ring $\mathcal{PK}$. So, σ is a forgery. If $\mathcal{F}_{\mathsf{rvrf}}$ is in Cond.- 1 and sets $b = 0$, it means that $m, W, \mathcal{PK}, \sigma$ was recorded as invalid by $\mathcal{F}_{\mathsf{rvrf}}$, but now $m, W, \mathcal{PK}, \sigma$ is a valid signature in the real world. This case never happens because of the correctness of NIZK algorithms.
  - If $b = b_{\mathsf{Sim}}$, set $\texttt{oracle\_queries\_h}[m, \mathcal{PK}, W] = y$. Here, if σ is a signature of an honest party, Sim sets its output with respect to the output selected by $\mathcal{F}_{\mathsf{rvrf}}$.
- If $b_{\mathsf{Sim}} = 0$, it sets $X = \perp$ and sends $(\mathsf{verified}, \mathsf{sid}, \mathcal{PK}, W, m, \sigma, b_{\mathsf{Sim}}, X)$ to $\mathcal{F}_{\mathsf{rvrf}}$. Then, Sim receives back $(\mathsf{verified}, \mathsf{sid}, \mathcal{PK}, W, m, \sigma, y, b)$.
  - If $b \neq b_{\mathsf{Sim}}$, it means that it was a signature of an honest party and NIZK.Verify for $\mathcal{R}_{\mathsf{com}}$ does not validate in the real world. So, Sim sets $\texttt{oracle\_queries\_h}[m, \mathcal{PK}, W] = y$ and $\texttt{oracle\_queries\_h\_schnor}[\mathcal{PK}, m, W, C, R', R'_m] = c$ where $R' = sG_1 + \delta G_2 + cC$, $R_m = s\mathsf{H}_{\mathbb{G}}(m, \mathcal{PK}) + cW$. Now, the signature verifies in the real world as well.
  - If $b = b_{\mathsf{Sim}}$, Sim doesn't need to do anything.

– **[Simulation of link:]** Upon receiving $(\mathsf{link}, \mathsf{sid}, \mathcal{PK}, W, X_i, m, \sigma)$ from $\mathcal{F}_{\mathsf{rvrf}}$, Sim runs $\mathsf{NIZK.Simulate}(\mathcal{R}_{dleq}, (G_1, \mathbb{G}, X_i, P, W))$ and obtains $\hat{\sigma}$.
– **[Simulation of verifylink:]** Upon receiving $(\mathsf{verifylink}, \mathsf{sid}, \mathsf{pk}, \mathcal{PK}, W, m, \sigma, \hat{\sigma})$ from the functionality $\mathcal{F}_{\mathsf{rvrf}}$, it runs $\mathsf{rVRF.LinkVerify}(\mathsf{pk}, \mathcal{PK}, m, \sigma, \hat{\sigma}) \rightarrow b_{\mathsf{Sim}}$ and returns $(\mathsf{verified}, \mathsf{sid}, \mathsf{pk}, \mathcal{PK}, m, \sigma, \hat{\sigma}, b_{\mathsf{Sim}})$ to $\mathcal{F}_{\mathsf{rvrf}}$. Then, $\mathcal{F}_{\mathsf{rvrf}}$ replies with $(\mathsf{verified}, \mathsf{sid}, \mathsf{pk}, \mathcal{PK}, W, m, \sigma, \hat{\sigma}, b)$.
If $b' \neq b_{\mathsf{Sim}} = 1$ and $\mathsf{pk} \in \texttt{honest\_verification\_keys}$, then Sim aborts. If this happens, it means that a link signature for σ has not been yet generated for $m, \mathcal{PK}$ by an honest party with a key pk in the ideal world.
We remark that the other cases: $(b' \neq b_{\mathsf{Sim}} = 1, \mathsf{pk} \in \texttt{malicious\_verification\_keys})$ and $b \neq b_{\mathsf{Sim}} = 0$ cannot happen thanks to the correctness of NIZK. The first case means that

$\mathsf{rVRF.LinkVerify}(\mathsf{pk}, \mathcal{PK}, m, \sigma, \hat{\sigma}) \rightarrow 0$ before, but now $\mathsf{rVRF.LinkVerify}(\mathsf{pk}, \mathcal{PK}, m, \sigma, \hat{\sigma}) \rightarrow 1$. The second case $b \neq b_{\mathsf{Sim}} = 0$ is not possible also because if pk is an honest key, its link signature is generated by Sim so $\mathsf{rVRF.LinkVerify}$ always verifies it . If pk is a malicious key then $\mathcal{F}_{\mathsf{rvrf}}$ outputs b_{Sim} if it was not set 0 before.

Theorem 2. *rVRF over the group structure $(\mathbb{G}, p, G_1, G_2)$ realizes $\mathcal{F}_{\mathsf{rvrf}}$ in Figure 2 in the random oracle model assuming that NIZK is zero-knowledge and knowledge extractable, the decisional Diffie-Hellman (DDH) problem are hard in $(\mathbb{G}, p, G_1, G_2)$.*

Proof. We first show that the outputs of honest parties in the ideal world are indistinguishable from the output of honest parties in the real protocol.

Lemma 1. *Assuming that DDH problem is hard on the group structure $(\mathbb{G}, G_1, G_2, p)$, 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 link signatures and outputs of the ring signatures in the ideal world identical to the real world protocol because the link signature is generated by Sim as in the real protocol and the outputs are randomly selected by $\mathcal{F}_{\mathsf{rvrf}}$ as the random oracle H in the real protocol. The only difference is the generation of the ring signature (See Algorithm 2) and the anonymous key (See Algorithm 1). The distribution of $\pi_{com} = (c, s, \delta)$ and C generated by Algorithm 2 and the distribution of $\pi_{com} = (c, s, \delta)$ and C generated by $\mathsf{rVRF.Sign}$ are from uniform distribution so they are indistinguishable. Thanks to the zero-knowledge property of NIZK, π_{MT} generated by Algorithm 2 and π_{MT} generated by $\mathsf{rVRF.Sign}$ are indistinguishable too.

Now, we show that the anonymous key W generated by Algorithm 1 and W generated by $\mathsf{rVRF.Sign}$ are indistinguishable. For this, we need show that selecting W randomly from $\mathbb{G}$ and computing W as $x\mathsf{H}_{\mathbb{G}}(m, \mathcal{PK})$ 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 $X, Y, Z \in \mathbb{G}$ 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 $\mathbb{G}, G_1 = Y, G_2$ 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 2 and Algorithm 1. It generates the link signatures of $\mathsf{P}_{i+2}, \ldots, \mathsf{P}_{n_h}$ by running $\mathsf{NIZK.Simulate}(\mathcal{R}_{dleq}, (G_1, \mathbb{G}, X_i, P, W))$. 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 $m, \mathcal{PK}$, it obtains $P = \mathsf{H}_{\mathbb{G}}(m, \mathcal{PK}) = hY$ from `oracle_queries_gg` and lets $W = hZ$. Remark that if (X, Y, Z) is a DH triple (i.e., $\mathsf{DH}(X, Y, Z) \rightarrow 1$), P_{i+1} is simulated as in rVRF because $W = xP$ in this case. Otherwise, P_{i+1} is simulated as in the ideal world because W is random. So, if $\mathsf{DH}(X, Y, Z) \rightarrow 1$, 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 $\mathsf{DH}(X, Y, Z) \rightarrow 0$. 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 2. *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.*

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 1 that the output of honest parties are indistinguishable from the real protocol. Next, we show that the abort cases happen with negligible probability during the simulation. In other words, 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 the group prime p-order group $\mathbb{G}$ with the challenges $G_1, U, V \in \mathbb{G}$. The CDH challenges are given to the simulator $\mathcal{B}$. Then $\mathcal{B}$ runs a simulated copy of $\mathcal{Z}$ and starts to simulate $\mathcal{F}_{\mathsf{rvrf}}$ and Sim for $\mathcal{Z}$. For this, it first runs the simulated copy of $\mathcal{A}$ as Sim does. $\mathcal{B}$ provides $(\mathbb{G}, p, G_1, G_2)$ as a public parameter of the ring VRF protocol to $\mathcal{A}$.

Whenever $\mathcal{B}$ needs to generate a ring signature for $m, \mathcal{PK}$ on behalf of an honest party with a public key X, it runs Algorithm 3 to generate the corresponding anonymous key of the honest party and Algorithm 4.

Algorithm 3 $\mathsf{Gen}_W(\mathcal{PK}, X, m)$

if $DB_W[m, \mathcal{PK}, X] = \perp$ **then**
  $W \leftarrow\!\!\$\ \mathbb{G}$
  $DB_W[m, \mathcal{PK}, X] := \perp$
  add W to list $\mathcal{W}[m, \mathcal{PK}]$
return $DB_W[m, \mathcal{PK}, X]$

Algorithm 4 $\mathsf{Gen}_{sign}(\mathcal{PK}, W, X, m)$

$c, s, \delta \leftarrow\!\!\$\ \mathbb{F}_p$
$\beta \leftarrow\!\!\$\ \mathbb{F}_p$
$C = X + \beta G_2$
$R' = sG_1 + \delta G_2 + cC$
$R_m = s\mathsf{H}_{\mathbb{G}}(m, \mathcal{PK}) + cW$
$\texttt{oracle\_queries\_h\_schnor}[\mathcal{PK}, m, W, C, R', R'_m] = c$
$\pi_{com} \leftarrow (c, s, \delta)$
$\pi_{\mathsf{pk}} \leftarrow \mathsf{NIZK.Prove}(\mathcal{R}_{\mathsf{pk}}, ((X, \beta), (G_1, G_2, \mathbb{G}, \mathcal{PK}, C)))$
return $\sigma = (\pi_{com}, \pi_{MT}, C, W)$

Whenever $\mathcal{B}$ needs to generate a link signature for an honest party, it runs $\mathsf{NIZK.Simulate}(\mathcal{R}_{dleq}, (G_1, \mathbb{G}, X, P, W))$ as Sim does.

Clearly the ring signature of an honest party outputted by Sim (remember $\mathcal{F}_{\mathsf{rvrf}}$ generates it by Algorithm 2) and the ring signature generated by $\mathcal{B}$ are indistinguishable because NIZK is zero knowledge.

In order to generate the public keys of honest parties, $\mathcal{B}$ picks a random $r_x \in \mathbb{Z}_p$ 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 and link signatures without the secret keys. Therefore, generating the honest public keys in this way is indistinguishable.

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

Fig. 5. The random oracle H

Oracle $\mathsf{H}_{\mathbb{G}}$
Input: $m, \mathcal{PK}$
if $\texttt{oracle\_queries\_gg}[m, \mathcal{PK}] = \perp$
  $h \leftarrow\$ \mathbb{F}_p$
  $P \leftarrow hU$
  $\texttt{oracle\_queries\_gg}[m, \mathcal{PK}] := h$
else:
  $h \leftarrow \texttt{oracle\_queries\_gg}[m, \mathcal{PK}]$
  $P \leftarrow hU$
return P

Fig. 6. The random oracle $\mathsf{H}_{\mathbb{G}}$

Simulation of $\mathsf{H}_{\mathbb{G}}$ is as described in Figure 6 i.e., it returns hU instead of hG_1. The simulation of $\mathsf{H}_{\mathbb{G}}$ is indistinguishable from the simulation of $\mathsf{H}_{\mathbb{G}}$ in Figure 3. $\mathcal{B}$ 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 $\mathsf{H}(m, \mathcal{PK}, W)$ 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 $\mathsf{H}_{\mathbb{G}}$ is not simulated as in Figure 3, $\mathcal{B}$ cannot check whether W is an anonymous key generated by an honest key or not.

During the simulation whenever a valid signature $\sigma = (\pi_{com}, \pi_{\mathsf{pk}}, C, W)$ of message m signed by $\mathcal{PK}$ is outputted and $W \notin \mathcal{W}[m, \mathcal{PK}]$ (i.e., W is not generated by $\mathcal{B}$), Sim increments a counter $\mathsf{counter}[m, \mathcal{PK}]$ and adds W to $\mathcal{W}[m, \mathcal{PK}]$. Then it runs $\mathsf{Ext}(\mathcal{R}_{\mathsf{pk}}, \pi_{\mathsf{pk}_j}, (G_1, G_2, \mathbb{G}, \mathcal{PK}, C_j)) \rightarrow X_j, \beta_j$ where $X_j \in \mathcal{PK}$ and $C_j = X_j + \beta_j G_2$ and $\mathsf{Ext}(\mathcal{R}_{\mathsf{com}}, \pi_{com_j}, (G_1, G_2, \mathbb{G}, \mathcal{PK}, C_j, W_j, \mathsf{H}_{\mathbb{G}}(m, \mathcal{PK}))) \rightarrow (\hat{x}_j, \hat{\beta}_j)$ such that $C_j = \hat{x}_j G_1 + \hat{\beta}_j G_2$ and $W_j = \hat{x}_j \mathsf{H}_{\mathbb{G}}(m, \mathcal{PK})$.

If X_j is an honest public key and $X_j = \hat{x}_j G_1$, $\mathcal{B}$ solves the CDH problem as follows: $W = \hat{x}_j hU$ where $h = \texttt{oracle\_queries\_gg}[m, \mathcal{PK}]$. Since $X_j = r_j V$, $W = \hat{x}_j huG_1 = r_j huV$. So, $\mathcal{B}$ outputs $r_j^{-1} h^{-1} W$ as a CDH solution and simulation ends. Remark that this case happens when Sim aborts because of Cond.- 4.

If $\mathsf{counter}[m, \mathcal{PK}] = t \geq |\mathcal{PK}_{\mathcal{A}}|$, $\mathcal{B}$ obtains all the signatures $\{\sigma_i\}_{i=1}^{t}$ that make $\mathcal{B}$ increment $\mathsf{counter}[m, \mathcal{PK}]$ and solves the CDH problem as follows: Remark that this case happens when Sim aborts because of Cond.- 3.

For all $\sigma_j = (\pi_{com_j}, \pi_{\mathsf{pk}_j}, C_j, W_j) \in \{\sigma_i\}_{i=1}^{t}$, $\mathcal{B}$ runs $\mathsf{Ext}(\mathcal{R}_{\mathsf{pk}}, \pi_{\mathsf{pk}_j}, (G_1, G_2, \mathbb{G}, \mathcal{PK}, C_j)) \rightarrow X_j, \beta_j$ where $X_j \in \mathcal{PK}$ and $C_j = X_j + \beta_j G_2$. One of the following cases happens:

- All X_j's are different: If $\mathcal{B}$ is in this case, it means that there exists one public key X_a which is honest. Then $\mathcal{B}$ runs $\mathsf{Ext}(\mathcal{R}_{\mathsf{com}}, \pi_{com_a}, (G_1, G_2, \mathbb{G}, \mathcal{PK}, C_a, W_a, \mathsf{H}_{\mathbb{G}}(m, \mathcal{PK}))) \rightarrow (\hat{x}_a, \hat{\beta}_a)$ such that $C_a = \hat{x}_a G_1 + \hat{\beta}_a G_2$ and $W_a = \hat{x}_a \mathsf{H}_{\mathbb{G}}(m, \mathcal{PK})$. If $\mathcal{B}$ is in this case, $\hat{x}_a G_1 \neq X_a$ because otherwise it would solve the CDH as described before. Therefore, $\beta_a \neq \hat{\beta}_a$. Since $X_a + \beta_a G_2 = \hat{x}_a G_1 + \hat{\beta}_a G_2$ and $X_a = r_a V$ where r_a is generated by $\mathcal{B}$ during the key generation process, $\mathcal{B}$ obtains a representation of $V = \gamma G_1 + \delta G_2$ where $\gamma = \hat{x}_a r_a^{-1}$ and $\delta = (\hat{\beta}_a - \beta) r_a^{-1}$. 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., $V = \gamma' G_1 + \delta' G_2$. Thus, $\mathcal{B}$ can find discrete logarithm of V on base G_1 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 where $X_a = X_b$. $\mathcal{B}$ runs $\mathsf{Ext}(\mathcal{R}_{\mathsf{com}}, \pi_{com_a}, (G_1, G_2, \mathbb{G}, \mathcal{PK}, C_a, W_a, \mathsf{H}_{\mathbb{G}}(m, \mathcal{PK}))) \rightarrow (\hat{x}_a, \hat{\beta}_a)$ and $\mathsf{Ext}(\mathcal{R}_{\mathsf{com}}, \pi_{com_b}, (G_1, G_2, \mathbb{G}, \mathcal{PK}, C_b, W_b, \mathsf{H}_{\mathbb{G}}(m, \mathcal{PK}))) \rightarrow (\hat{x}_b, \hat{\beta}_b)$ such that $C_a = \hat{x}_a G_1 + \hat{\beta}_a G_2, C_b = \hat{x}_b G_1 + \hat{\beta}_b G_2$ and $W_a = \hat{x}_a \mathsf{H}_{\mathbb{G}}(m, \mathcal{PK}), W_b = \hat{x}_b \mathsf{H}_{\mathbb{G}}(m, \mathcal{PK})$. 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., $X_a = X_b = \hat{x}_a G_1 + (\hat{\beta}_a - \beta_a) G_2 = \hat{x}_b G_1 + (\hat{\beta}_b - \beta_b) G_2$. Thus, $\mathcal{B}$ finds the discrete logarithm of $G_2 = U$ in base G_1 which is $u = \frac{\hat{x}_a - \hat{x}_b}{\hat{\beta}_a - \beta_a - \hat{\beta}_b + \beta_b}$. $\mathcal{B}$ 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, then we can construct an adversary $\mathcal{B}$ that solves the CDH problem except with a negligible probability.

□

6 Zero-knowledge continuations

We now construct ring VRFs which achieves fast amortized prover time by using a heavy zero-knowledge continuation for $\mathsf{rVRF.OpenRing}$ but which permits updating oppk in the $\mathsf{PedVRF.OpenKey}$ invocation without reproving $\pi_{\mathtt{ring}}$.

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

6.1 Rerandomization

Zero-knowledge continuations need rerandomizable zkSNARKs when being reused multiple times, meaning Groth16 [6], but unlinkability requires more than merely rerandomization.

In Groth16 [6], we have an SRS S consisting of curve points in $\mathbf{E}_1$ and $\mathbf{E}_2$ that encode the circuit being proven. We follow [6] 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 $[\cdots]_2$ to denote an element of our SRS S in $\mathbf{E}_1$ or $\mathbf{E}_2$ respectively, computed by scalar multiplication 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 $\chi_1, \ldots, \chi_k, [\alpha]_1 \in \mathbf{E}_1$ and $[\beta]_2, [\gamma]_2, [\delta]_2 \in \mathbf{E}_2$. A Groth16 [6] proof then takes the form $\pi = (A, B, C) \in \mathbf{E}_1 \times \mathbf{E}_2 \times \mathbf{E}_1$. A verifier then produces a $X = \sum_i^k x_i \chi_i \in \mathbf{E}_1$ 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 [1, 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 also need one fresh basepoint K_γ independent from all others, again perhaps created by applying $H_{\mathbf{E}}$ to an input outside existing usages' domain. We now give provers the additional SRS elements

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

Although K_γ is independent, we create K_δ during the trusted setup, so the toxic waste γ and δ remain secret. After this, subversion resistance could be checked like

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

We now have a zero-knowledge continuation (X, A, B, C) from which we produce an unlinkable instance (X', A', B', C') by first sampling random $b, r_1, r_2 \in \mathbb{F}_p$ and then computing

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

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

Along side hidden inputs in $X = \sum_i^k x_i \chi_i$, our verifier would typically assemble some transparent inputs $Y = \sum_i^l y_i \Upsilon$ themselves. In particular, our ring VRFs handle the Merkle root $\mathsf{comring}$ of the ring ctx in this way below.

We then verify $(X' + Y, A', B', C')$ like Groth16

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

As our verifier does not build X' themselves, we proves nothing with this pairing equation unless the verifier separately checks some proof-of-knowledge that $X' = \sum_i^k x_i \chi_i$.

Lemma 3. *Our rerandomization procedure transforms honestly generated zero-knowledge continuations (X, A, B, C) into identically distributed zero-knowledge continuations (X', A', B', C'), with identical transparent inputs $y_1, \ldots, y_l$.*

Proof (Proof idea.). Adapt the proof of Theorem 3 of [1, 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 Y vs opaque unlinkable part X.

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

Lemma 4. *Assuming AGM plus the $(2n-1, n-1)$-DLOG assumption, any zero-knowledge continuation with circuit size less than n satisfies knowledge soundness, restricted to challengers who learn the actual public input wire values and blinding factors.*

Proof (Proof idea.). Adapt the proof of Theorem 2 in [1, §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 under similar restrictions, much like Theorem 1 in [1, §3, pp. 8 & 11]. We depends upon the specific simulator though, which itself increases our dependence upon the usage of the zero knowledge continuation.

6.2 Single continuation

We describe a much faster choice $\pi_{\mathtt{fast}}$ for π that sets $x_1 := \mathsf{sk}$ and $x_0 = \mathsf{comring} = \mathsf{CommitRing}(\mathsf{ctx})$ so that taking $G := \chi_1$, $K := K_\gamma$, and $\mathsf{oppk} := b$ in PedVRFyields and incredibly fast amortized ring VRF prover. Also, PedVRFitself proves knowledge of X', or more precisely of $X' - \mathsf{comring}\,\chi_0$ since the verifier knows $\mathsf{comring}$.

$$X' = \mathsf{comring}\,\chi_0 + \mathsf{sk}\,\chi_1 + bK_\gamma$$

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

We know the large prime order group $\mathbf{J}$ of $\mathbb{J}$ typically has smaller order than $\mathbf{E}$, itself due to $\mathbb{J}$ being an Edwards curve. Yet, if $\mathsf{sk} = \mathsf{sk}_0 + \mathsf{sk}_1\,2^{128}$ then our public key commitments could take the form $\mathsf{sk}_0\,J_0 + \mathsf{sk}_1\,J_1 + dJ_2$, with independent J_0, J_1, J_2. Interestingly, we avoid range proofs for sk_1 and sk_2 by this independence.

$$\pi_{\mathtt{fast}} = \mathsf{Groth16}\left\{ \mathsf{sk}_0 + \mathsf{sk}_1 2^{128}, \mathsf{comring} \;\middle|\; \exists d, \mathsf{opring}\ \text{s.t.}\ \begin{array}{r} \mathsf{rVRF.OpenRing}(\mathsf{comring}, \mathsf{opring}) \\ = \mathsf{sk}_0 J_0 + \mathsf{sk}_1 J_1 + dJ_2 \end{array} \right\}$$

Applying our rerandomization to $\pi_{\mathtt{fast}}$ with opaque input yields a zkSNARK $\pi'_{\mathtt{fast}}$ of exactly the form $\pi_{\mathtt{ring}}$.

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

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

As an aside, one could construct a second faster curve with the same group order as **E**, 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 3. *Our Pedersen ring VRF instantiated with $\pi_{\texttt{fast}}$ satisfies knowledge soundness.*

Proof (Proof stetch.). An extractor for PedVRFreveals the opening of X for us, so our result follows from Lemma 4.

6.3 Side channel

In this, we dislike exposing the secret key material inside the Groth16 prover for $\pi_{\texttt{fast}}$. Adversaries could trigger $\pi_{\texttt{fast}}$ 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 $\pi_{\texttt{fast}}$ into two parts:

$$\pi_{\mathsf{pk}} = \mathsf{Groth16/KZG}\left\{ J_{\mathsf{pk}}, \mathsf{comring} \mid \exists \mathsf{opring} \text{ s.t. } J_{\mathsf{pk}} = \mathsf{rVRF.OpenRing}(\mathsf{comring}, \mathsf{opring}) \right\}. \quad \text{and}$$

$$\pi_{\mathsf{sk}} = \mathsf{Groth16}\left\{ \mathsf{sk}_0 + \mathsf{sk}_1 2^{128}, J_{\mathsf{pk}} \;\middle|\; \exists d \text{ s.t. } J_{\mathsf{pk}} = \mathsf{sk}_0 J_0 + \mathsf{sk}_1 J_1 + d J_2 \right\}.$$

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

There is no "arrow of time" among zk continuations per se, but as π_{sk} bridges between the PedVRFand π_{pk}, one might consider this zk continuation to be time reversed.

Interestingly, our π_{pk} has become simple enough that opening a KZG commitment [7] at a secret point now suffices, ala Caulk+ [10], or perhaps Caulk [11].

We could build our ring using a blockchain, not unlike how zcash handles their UTXOs. If so, we suggest $\pi_{\texttt{fast}}$ provide an inner per block or per epoch ring, but then another proving system like Caulk+ [10] extend this ring to the growing blockchain, like

$$\pi_{\texttt{chain}} = \mathsf{Groth16/KZG}\left\{ \mathsf{comblock}, \mathsf{comring} \mid \exists \mathsf{opring} \text{ s.t. } J_{\mathsf{pk}} = \mathsf{rVRF.OpenRing}(\mathsf{comring}, \mathsf{opring}) \right\}. \quad \text{and}$$

$$\pi_{\texttt{fast}} = \mathsf{Groth16}\left\{ \mathsf{comblock}, \mathsf{comring} \;\middle|\; \exists d, \mathsf{opring} \text{ s.t. } \begin{array}{c} \mathsf{rVRF.OpenRing}(\mathsf{comblock}, \mathsf{opring}) \\ = \mathsf{sk}_0 J_0 + \mathsf{sk}_1 J_1 + d J_2 \end{array} \right\}.$$

Again we need a proof=of-knowledge for comblock because it winds up opaque in both of these proofs.

A priori, our JubJub representations $\mathsf{sk}_0 J_0 + \mathsf{sk}_1 J_1$ costs us exculpability from Definition **??**. If desired, exculpability could easily be repaired by asking users provide π_{sk} when joining the ring.

A Secret Ring VRF

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

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

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

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

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

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

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

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

References

1. 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.
2. 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.
3. Privacy by Design Foundation. Irma docs v0.2.0. https://irma.app/docs/v0.2.0/overview/.
4. 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.
5. 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.
6. 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.
7. 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.

8. 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.
9. 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/.
10. Jim Posen and Assimakis A. Kattis. Caulk+: Table-independent lookup arguments. Cryptology ePrint Archive, Paper 2022/957, 2022. https://eprint.iacr.org/2022/957.
11. 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.