SyRA: Sybil-Resilient Anonymous Signatures with Applications to Decentralized Identity

Elizabeth Crites[1], Aggelos Kiayias[2], and Amirreza Sarencheh[⋆2]

[1] Web3 Foundation, elizabeth@web3.foundation
[2] The University of Edinburgh & IOG, UK, firstname.lastname@ed.ac.uk

Abstract. We introduce a new cryptographic primitive, called Sybil-Resilient Anonymous (SyRA) signature, which enables users to generate, on demand, unlinkable pseudonyms tied to any given context, and issue digital signatures on their behalf. Concretely, given a *personhood relation*, an issuer (who may be a distributed entity) enables users to prove their personhood and extract an associated long-term key, which can then be used to issue signatures for any given context and message. Sybil-resilient anonymous signatures achieve two key security properties: 1) *Sybil resilience*: ensures that every user is entitled to *at most one* pseudonym per context, and 2) *anonymity*: requires that no information about the user is leaked through their various pseudonyms or the signatures they issue on their pseudonyms' behalf.

We conceptualize SyRA signatures as an ideal functionality in the Universal Composition (UC) setting and realize the functionality via an efficient, pairing-based construction that utilizes two levels of verifiable random functions (VRFs) and which may be of independent interest. A key feature of this approach is the statelessness of the issuer: we achieve the core properties of Sybil resilience and anonymity without requiring the issuer to retain *any* information about past user interactions.

SyRA signatures have various applications in multiparty systems, such as e-voting (e.g., for decentralized governance) and cryptocurrency airdrops, making them an attractive option for deployment in decentralized identity (DID) systems.

Keywords: Decentralized Identity, Digital Identity, Privacy, Sybil Resilience, Decentralized Governance, Universal Composition, Digital Signature

1 Introduction

A recent development in identity systems is the concept of decentralized identity (DID) which comes with the promise of empowering users by enabling them to be "self sovereign" in terms of identity management.[3] This means that users

⋆ Corresponding author.

[3] The W3C Decentralized Identifier working group [Con22] and Decentralized Identity Foundation [Fou23] have been developing standards for DIDs.

have the ability to self manage the secret key(s) behind their identity and access services that require identification without a trusted third party acting as an intermediary, as is the case with single-sign on systems, cf. [WCW12,FKS16]. Given that users may be operating across a multitude of *contexts*, it is important for them to be able to generate "context-specific" credentials that, for privacy considerations, must be unlinkable.[4] From a cryptographic perspective, these considerations suggest the following two seemingly contradictory objectives for realizing DIDs in a privacy-preserving manner.

- *Sybil Resilience.* With the help of the issuing authority, each user can translate their real-world identity into a master credential, which can then be mapped on demand and noninteractively to *at most one* context-specific credential for any given context.
- *Anonymity.* Context-specific credentials reveal nothing of the user's identity, allowing users to be unlinkable across different contexts.

It is easy to see that the above properties can be trivially achieved individually: without privacy, Sybil resilience is straightforward to achieve from standard credentials, as the user's identity can follow them across different contexts. On the other hand, without Sybil resilience, anonymity can be achieved by standard cryptographic techniques, such as blind signatures [Cha81].

But how is it possible to achieve both properties simultaneously? A first approach can be derived from unclonable group identification [DDP06,CHK+06], where context is restricted to be in the form of distinct periods, and users are capable of authenticating themselves in a Sybil-resilient and unlinkable manner. In these constructions, the issuer verifies the real-world identity of the user and maps it to a credential that has the desired Sybil-resilient context-specific property across periods. Still, it is far more desirable for real-world use cases to be able to use as context any given string; this amounts to going from a polynomial-size context space to an exponential one. The issue of a large space context can be addressed via the concept of traceable ring signatures [FS07], where signing twice for the same "issue" identifies the public key of the signer.

Nevertheless, an important consideration remains: in all these previous approaches, the connection to the real-world identity of the signer is tenuous – it fundamentally relies on the ability of the credential issuer to maintain a mapping between credentials and real-world identities. This raises scalability, security, and privacy considerations, namely: how is it possible to manage an ever-growing such mapping of real-world identities to credentials, distribute such a mapping across many servers with Byzantine fault tolerance, and, at the same time, keep it private in order to prevent adversaries from performing deanonymization attacks via correlation? We refer to these considerations as the *(credential) issuer state problem.*

Solving the issuer state problem poses a conundrum that begs cryptographic treatment: Sybil resilience seems to mandate that the issuer must maintain a

[4] See Sec. 4.2 of the Peer-DID standard that covers-context specific DIDs for users https://identity.foundation/peer-did-method-spec/index.html.

mapping of real-world identities to credentials; indeed, a failure to do so would permit a trivial attack against Sybil resilience: the user would exploit the confusion of the issuer to dispense more than one credential for itself and thus become a Sybil. On the other hand, scaling and distributing the issuer would benefit from obfuscating, or even eliminating (if possible!), such state altogether.

Motivated by the connection of all of the above to the DID problem, in this work we focus on the following question: *What is a natural cryptographic primitive that reconciles Sybil Resilience with Anonymity, and how can we realize it in a way that solves the issuer state problem?*

Our Results. We tackle the above question by putting forth a new primitive, Sybil-Resilient Anonymous (SyRA) signatures, and realizing it via an efficient construction that facilitates a *stateless* distributed issuer entity.

In a SyRA signature scheme, an issuing authority, that may be distributed, is tasked with translating a real-world identity s into a signing key for the prospective user.[5] The issuing authority introduces a user into the system whose identity string satisfies a public *personhood relation* R and must ensure that there is a one-to-one correspondence between issued keys and distinct, real-world identities. Once the issuing protocol succeeds, a SyRA signature scheme enables a user to sign messages for any given context ctx; signatures can be verified against the public key of the issuing authority and the context string. The signing procedure generates a context-specific pseudonym, or tag T, that attaches itself to each signature. This enables the user to create separate and unlinkable pseudonyms to issue signatures across different contexts. Nevertheless, within a specific context, the signatures issued by the user can be linked to each other.

To capture formally the properties of this new primitive, we provide an ideal functionality for SyRA signatures in Section 3. Some key characteristics of this functionality are as follows. First, it verifies that the prospective user possesses a valid witness for their identity s with respect to the personhood relation. Second, upon each signature generation step, it creates a unique user-context identifier that is leaked to the adversary. This ensures that as long as unique user-context combinations are being exercised, there is no loss of anonymity. On the other hand, once the same user signs with the same context twice, the adversary will be notified by receiving the same user-context identifier. Note that this does not compromise the privacy of other contexts signed by the same user. Finally, the verification interface enables the verifier to publicly extract the pseudonym of each signature.

Armed with our ideal functionality, we set out to provide a construction for SyRA signatures in Section 4. From a design perspective, the challenge is to embed the identity of the user into their signing key so that signatures are unlinkable as long as distinct contexts are signed by different users, and at the same time enable detection for user signatures on the same context. Crucially,

[5] Note that for SyRA signatures with distributed issuance, we cannot expect to have privacy if all issuers are malicious, since in that case, they could always impersonate a real-world user with identity s and use the tracing operation to find all of the user's signatures.

we cannot rely on any auxiliary cryptographic structure in the membership credential – this is because even if the user repeats the whole issuing protocol again, as long as their real-world identity s remains the same, the user should not be able to sign the same context more than once without being identified. In this way, our issuer can be stateless.

Our construction overcomes this difficulty via the composition of two verifiable random functions (VRF). The first VRF is keyed by the issuing authority. During issuance, the user obtains the VRF value on their real-world identity string s while proving to the issuing authority that the personhood relation R is satisfied. In this manner, the user obtains a value that is deterministically tied to their real-world identity s. The second challenge is to ensure that this value can be safely used as the cryptographic key to a second VRF. The reason we aim for a second VRF here is to ensure unlinkability on the one hand while still linking by pseudonym on the other, for all signatures issued in the same context.

For ease of presentation, we present our construction in two steps. First, we consider a central issuer that is trusted for privacy, but with the constraint that it is *stateless*, i.e., the issuer should achieve Sybil resilience *without maintaining* in their state the set of real-world identities (or anything that is derived from them) that have been served SyRA credentials. We then demonstrate how to distribute the issuer. In the construction, we employ the Dodis-Yampolskiy verifiable random function (VRF) [DY05], which we extend to Type-III bilinear groups. Our asymmetric DY VRF produces the same classical DY VRF output in the target group as well as a proof of correctness in one source group, but additionally outputs the same proof in the other source group. These two values, which have the form $(g^{1/(s+\mathsf{isk})}, \hat{g}^{1/(s+\mathsf{isk})})$ serve as the user secret key in our SyRA signature scheme, computed by the issuer holding the issuing secret key isk on the user's identity string s.

The second primitive in our construction is inspired by the verifiable unpredictable function (VUF) of [GJM+21], which has the unique property that it employs a *group* element secret key. This is a key ingredient in our SyRA signature scheme, as the output of the first VRF becomes the secret key of the second. The VUF of [GJM+21] leveraged the determinism of BLS signatures [BLS04] and a Groth-Escala NIZK [EG14] style proof of correctness with respect to the group element key. Indeed, the output of the VUF is the pairing $T = e(\mathsf{h}(\mathsf{ctx}), \hat{\mathsf{usk}})$, where $\hat{\mathsf{usk}}$ is part of the user secret key, ctx is the context, and h is a hash function modelled as a random oracle. Our Sybil-resilient signatures have the same output, but we swap the Groth-Sahai NIZK proof of correctness with a Sigma protocol, as Groth-Sahai proofs do not support the zero-knowledge property for statements that involve the target group. Furthermore, we show in the random oracle model that $T = e(\mathsf{h}(\mathsf{ctx}), \hat{\mathsf{usk}})$ is not only unpredictable, but actually pseudorandom. It follows that the second primitive is also a VRF, which is essential for our privacy property.

We proceed to "thresholdize" the issuer by enabling multiple parties, each in possession of a share of the issuer secret key, to jointly compute a secret key pair, as above, for each user in the system. Distributed generation of user se-

cret keys enhances security and the availability of keying material. In the case of identity management, any cryptographic primitive used by certification authorities to issue credentials makes for an especially attractive target for misuse or forgery. Furthermore, a user's identity string s may be private or sensitive information. Unlike the centralized construction, no single issuer ever sees the user's identity string in the clear, and a threshold number of them would need to collude in order to break privacy. As claimed in the original Dodis-Yampolskiy paper, it is straightforward to construct a distributed computation of the function $F_{\mathsf{isk}}(s) = g^{1/(s+\mathsf{isk})}$ when the issuers have shares of the secret isk, and indeed we realize this using standard techniques for multi-party addition, inversion, and exponentiation [Bea92,KPR18].

We conclude our exposition with an overview of deployment considerations and applications for SyRA signatures. In particular, we describe how the personhood relation may be realized in an actual deployment, and how SyRA signatures apply to various use cases, such as e-voting, cryptocurrency airdrops, and solving the peer DID objective, cf. Section 7.

Related Work. Below we overview prior cryptographic primitives and their relation to SyRA signatures and our construction.

Group Signatures. Group signatures, introduced by Chaum and van Heyst [Cv91], allow a member of a group to anonymously sign messages on behalf of the group. The main security properties of group signatures are (1) unforgeability: only group members can create valid signatures; (2) anonymity: message-signature pairs do not leak the identity of the signer; (3) unlinkability: two message-signature pairs cannot be linked to the same signer; and (4) opening: the group manager can determine the identity of any signer using a special trapdoor. Similar to group signatures, in *traceable* signatures each group member has a tracing token that, if revealed, can link signatures from the same signer [KTY04]. In SyRA signatures, there is no opening (or tracing) authority; however, each signature incorporates an obfuscation of the user's identity string s in such a way that the user is unlinkable across different contexts, but entirely linkable within the same context.

Ring Signatures. Ring signatures, introduced by Rivest, Shamir, and Tauman Kalai [RST01], allow a member of an group, called a ring, to anonymously sign messages on behalf of the group. Ring signatures are similar to group signatures; however, there is no group manager who creates keys and manages the group. Instead, the formation is ad hoc. Ring signatures provide a strong notion of privacy: there is no mechanism for de-anonymizing users from the signatures they produce.

Identity-Based Signatures. Identity-based signatures (IBS) [Sha84] allow a user's signatures to be associated with a real-world identity string, such as an email address or phone number. User secret keys are generated by a master entity who ties them to identities using secret information. [Her06] proposed a *deterministic* identity-based signature scheme, where the signing is deterministic, i.e., signing the same message twice results in the same signature. [Hes03] proposed an IBS where instead the key issuance phase is deterministic. Thus, a user can-

not obtain multiple secret keys associated with their identity string. In a *hidden* identity-based signature scheme [KZ07], the user's identity string is not public information. SyRA signatures also facilitate hiding identities as in [KZ07], but offer linkability for multiple signatures on the same context.

Anonymous Credentials. The concept of using context-specific pseudonyms appears first in the context of anonymous credentials, beginning with the work of Chaum [Cha85] and [CE86], and followed up by subsequent works, cf. [CL01]. In this setting, users have distinct and unlinkable pseudonyms across organizations who play an active role in engaging with users transferring their credentials between them. In SyRA signatures, organizations can be entirely passive, each one defining their own set of context strings and otherwise allowing users to create and authorize their pseudonym for the contexts that are of interest to them.

Unclonable group identification. As discussed above, this line of work, beginning with [DDP06,CHK+06], covers a setting in which users are unlinkable across discrete time periods – a more limited setting compared to arbitrary context strings. [CHK+06] builds on prior work on e-cash [CHL05,CHL06], with the addition of this type of context.

Linkable and Traceable Ring Signatures. Linkable ring signatures [LWW04] allow for two signatures issued by the same signer with respect to the same ring of keys to be linked. *Traceable* ring signatures [FS07,ALSY13] generalize this notion to arbitrary contexts and allow for the recovery of the public key of the signer in case of signing twice in the same context. *One-time traceable* ring signatures [SZ21] facilitate the recovery of the public key of the signer in the linkable ring signature scenario. A similar type of functionality is achieved in the context of e-petitions in [DKD+08], where users can only sign a petition once. As mentioned earlier, in all these primitives, the connection of the user's key to their real-world identity is via an explicit mapping maintained by the issuer – while in principle such techniques can be used to produce a SyRA construction, they fundamentally rely on an *issuer maintained* mapping of real-world identities to keys.

Legacy compatible identification. In this setting, put forth in [ZMM+20] and utilized in the context of DIDs in CanDID [MMZ+21], the users can prove a piece of information, that can be their real-world identity, from a "legacy" web-service to a third-party provider. In the context of CanDID, such a provider (which is in fact a distributed entity) can issue a credential that the user can on demand (albeit interactively with the issuer) transpose to any context he wishes, while remaining unlinkable; furthermore, Sybil resilience is ensured with respect to the user's real-world identity. The CanDID construction makes the only attempt (to the best of our knowledge) towards solving the issuer state problem: the real-world identity of the user is obfuscated in the form of its PRF evaluation via a key that is shared between the issuing authorities. Each issuing authority keeps a copy of this obfuscated database and, hence, any attempt of the user to subvert Sybil resilience can be blocked by checking the PRF evaluation of their real-world identity against the obfuscated database. It is worth noting that this approach still has the downsides that each issuer needs to maintain a state linear in the number of credentials, and that users who lose their keys need

additional infrastructure to help them recover it, as the issuers, in their effort to protect against Sybils, will deny re-issuance to any user who is unfortunate enough to lose their key. Our SyRA construction, on the other hand, circumvents both of these downsides by requiring no (updateable) state for the issuers as well as it does not require interaction to introduce users to new contexts.

zkLogin [BCJ+24] similarly allows users to prove pieces of information from legacy web-services, but specifically in order to authorize cryptocurrency transactions. In particular, it leverages legacy providers in the OpenID Connect network, which include Google, Facebook, many other major platforms. Such a provider maps real-world identities to keys in such a way that it obviates the need for users to self-manage signing keys associated with digital wallets. It essentially acts like an identity-based signature scheme, where the OpenID provider is the key distribution authority. As in IBS, the real-world identity being used is the user's email address. A similar approach is taken by ZK Address Abstraction [PLL+23], i.e., using a zero-knowledge proof over the signed statement issued by the provider on user information in order to authenticate blockchain transactions. While these schemes protect the private data of users, neither addresses the property of Sybil resilience.

2 Preliminaries

Let $k \in \mathbb{N}$ denote the security parameter and 1^k its unary representation. Let p be a k-bit prime. For all positive polynomials $f(k)$, a function $\nu : \mathbb{N} \to \mathbb{R}^+$ is called *negligible* if $\exists\ k_0 \in \mathbb{N}$ such that $\forall\ k > k_0$ it holds that $\nu(k) < 1/f(k)$. We denote by $\mathbb{G}^*$ the set $\mathbb{G} \setminus 1_{\mathbb{G}}$, where $1_{\mathbb{G}}$ is the identity element of the group $\mathbb{G}$. We denote the group of integers mod p by $\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$ and its multiplicative group of units by $\mathbb{Z}_p^*$. We denote the set of integers $\{1, \ldots, n\}$ by $[1, n]$. Let $Y \xleftarrow{\$} F(X)$ denote running probabilistic algorithm F on input X and assigning its output to Y. Let $x \xleftarrow{\$} \mathbb{Z}_p$ denote sampling an element of $\mathbb{Z}_p$ uniformly at random. All algorithms are randomized unless expressly stated otherwise. PPT refers to probabilistic polynomial time.

Definition 1 (Bilinear Group). *A bilinear group generator* $\mathsf{GrGen}(1^k)$ *returns a tuple* $\mathsf{bp} \leftarrow (\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, p, e, g, \hat{g})$ *such that* $\mathbb{G}$, $\hat{\mathbb{G}}$ *and* $\mathbb{G}_T$ *are finite groups of the same prime order* p, $g \in \mathbb{G}$ *and* $\hat{g} \in \hat{\mathbb{G}}$ *are generators, and* $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$ *is a bilinear pairing, which is efficiently computable and satisfies (1) non-degeneracy:* $e(g, \hat{g}) \neq 1_{\mathbb{G}_T}$ *and (2) bilinearity:* $\forall\ x, y \in \mathbb{Z}_p$, $e(g^x, \hat{g}^y) = e(g, \hat{g})^{xy} = e(g^y, \hat{g}^x)$.

We rely on Type-III, or asymmetric, bilinear groups $\mathbb{G}$ and $\hat{\mathbb{G}}$, for which there is no efficiently computable isomorphism between them.

2.1 Cryptographic Assumptions

Assumption 1 (Decisional Diffie-Hellman (DDH)) *The decisional Diffie-Hellman assumption holds in* $\mathbb{G}$ *for* GrGen *if for all PPT adversaries* $\mathcal{A}$, *there*

exists a negligible function ν such that:

$$|\Pr[\mathcal{A}(\mathsf{bp}, g^x, g^y, g^r) = 1] - \Pr[\mathcal{A}(\mathsf{bp}, g^x, g^y, g^{xy}) = 1]| \leq \nu(k)$$

where in each case the probabilities are taken over the experiment in which $\mathsf{GrGen}(1^k)$ *outputs* $\mathsf{bp} = (p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g, \hat{g})$*, and then random* $x, y, r \in \mathbb{Z}_p$ *are chosen.*

The DDH assumption may be defined similarly for $\hat{\mathbb{G}}$ and $\mathbb{G}_T$.

Assumption 2 (q-Decisional Bilinear Diffie-Hellman Inversion (q-DBDHI)) [DY05] *Let* GrGen *be a group generator that outputs* $\mathsf{bp} = (\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, p, e, g, \hat{g})$*. The q-Decisional Bilinear Diffie-Hellman Inversion assumption holds for* GrGen *if for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* ν *such that:*

$$|\Pr[\mathcal{A}(\mathsf{bp}, g^x, \hat{g}^x, \ldots, g^{x^q}, \hat{g}^{x^q}; e(g, \hat{g})^{1/x})] - \Pr[\mathcal{A}(\mathsf{bp}, g^x, \hat{g}^x, \ldots, g^{x^q}, \hat{g}^{x^q}; \Gamma)]| \leq \nu(k)$$

where the probability is taken over the internal coin tosses of $\mathcal{A}$ *and choices of* $x \in \mathbb{Z}_p^*$ *and* $\Gamma \in \mathbb{G}_T$*.*

2.2 Verifiable Random Functions

A verifiable random function (VRF) is a function whose output is pseudorandom and for which a proof of correct evaluation is provided (i.e., a PRF with verifiability).

In our construction of a Sybil-resilient anonymous (SyRA) signature scheme, we employ a VRF to compute user secret keys. Recall the Dodis-Yampolskiy VRF in Fig. 3 [DY05]. It is defined over a *symmetric* pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, where the output is a value $F_{\mathsf{sk}}(\tau) = e(g, g)^{1/(\tau + \mathsf{sk})}$ in the target group, with corresponding proof $\pi_{\mathsf{vrf}} = g^{1/(\tau+\mathsf{sk})}$ in the source group. It is proven secure (i.e., pseudorandom) under the q-Decisional Bilinear Diffie-Hellman Inversion assumption (q-DBDHI) in [DY05]: Given $(g, g^x, \ldots, g^{x^q})$, it is hard to distinguish $e(g, g)^{1/x}$ from random. The q powers are used to simulate VRF output queries in the security reduction. In our SyRA signature scheme, we make use of the Dodis-Yampolskiy VRF in the *asymmetric* setting. In particular, user secret keys consist of a pair of group elements, one in each of the source groups, corresponding to a VRF proof of correctness in each group. We capture this in our definitions (Figs. 1 and 2) by allowing a second proof $\hat{\pi}_{\mathsf{vrf}}$ to be output. Pseudorandomness plays a key role in the proof of security for our construction (specifically, anonymity of users), and verifiability ensures users received correctly generated secret keys. Our asymmetric DY VRF is secure under the q-Decisional Diffie-Hellman inversion assumption (q-DBDHI) over asymmetric pairings (Assumption 2).

Definition 2 (Verifiable Random Function (VRF)). [DY05] *Let* $\ell : \mathbb{N} \to \mathbb{N} \cup \{*\}$ *and* $\ell' : \mathbb{N} \to \mathbb{N}$ *be any functions for which* $\ell(\kappa)$ *and* $\ell'(\kappa)$ *are computable in poly(*κ*) time (except when* ℓ *takes the value* $*$ *for an input of any length). A function family* $F_{(\cdot)}(\cdot) : \{0,1\}^{\ell(k)} \to \{0,1\}^{\ell'(k)}$ *is a family of VRFs if there exist polynomial-time algorithms* $(\mathsf{VRF.Setup}, \mathsf{VRF.Gen}, \mathsf{VRF.Prove}, \mathsf{VRF.Ver})$ *as follows:*

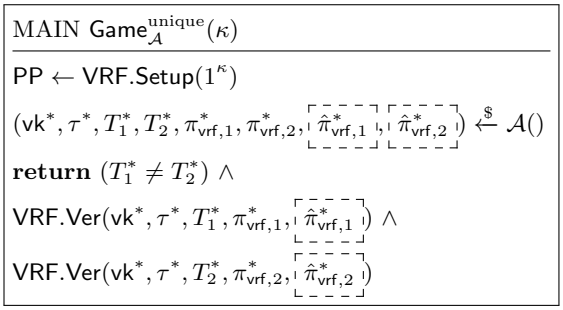

Fig. 1. Game used to define the uniqueness of a VRF (with optional second proof $\hat{\pi}_{\mathsf{vrf}}$).

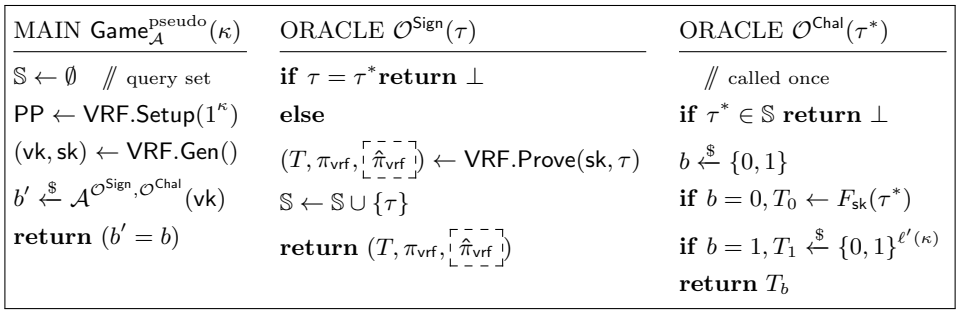

Fig. 2. Game used to define the pseudorandomness of a VRF (with optional second proof $\hat{\pi}_{\mathsf{vrf}}$).

$\mathsf{PP} \xleftarrow{\$} \mathsf{VRF.Setup}(1^\kappa)$: *a PPT algorithm that takes as input the security parameter and outputs public parameters* PP*, which are implicitly given as input to all other algorithms.*

$(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{VRF.Gen}()$*: a PPT algorithm that returns a public verification key* vk *and secret key* sk*.*

$(F_{\mathsf{sk}}(\tau), \pi_{\mathsf{vrf}}) \leftarrow \mathsf{VRF.Prove}(\mathsf{sk}, \tau)$*: an algorithm that takes as input a secret key and VRF input* τ *and returns a VRF output* $F_{\mathsf{sk}}(\tau)$ *and proof of correctness* π_{vrf}*.*

$0/1 \leftarrow \mathsf{VRF.Ver}(\mathsf{vk}, \tau, T, \pi_{\mathsf{vrf}})$*: a deterministic algorithm that takes as input a verification key, VRF input* τ*, VRF output* T*, and proof* π_{vrf} *and verifies that* $T = F_{\mathsf{sk}}(\tau)$ *using the proof* π_{vrf}*, outputting* 1 *to indicate acceptance or* 0 *to indicate rejection.*

The main security properties of a VRF are correctness, uniqueness, and pseudorandomness.

$\underline{\mathsf{VRF.Setup}(1^\kappa)}$

$\mathsf{bp} = (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathsf{GrGen}(1^\kappa)$

return $\mathsf{PP} = \mathsf{bp}$

$\underline{\mathsf{VRF.Prove}(\mathsf{sk}, \tau)}$

$T \leftarrow e(g,g)^{1/(\tau+\mathsf{sk})}$

$\pi_{\mathsf{vrf}} \leftarrow g^{1/(\tau+\mathsf{sk})}$

return (T, π_{vrf})

$\underline{\mathsf{VRF.Gen}()}$

$y \xleftarrow{\$} \mathbb{Z}_p^*;\ \mathsf{sk} \leftarrow y$

$\mathsf{vk} \leftarrow g^y$

return $(\mathsf{vk}, \mathsf{sk})$

$\underline{\mathsf{VRF.Ver}(\mathsf{vk}, \tau, T, \pi_{\mathsf{vrf}})}$

return $e(\pi_{\mathsf{vrf}}, g^\tau \cdot \mathsf{vk}) = e(g,g) \wedge$
$T = e(\pi_{\mathsf{vrf}}, g)$

Fig. 3. The Dodis-Yampolskiy VRF [DY05].

1. **Correctness.** For all $\kappa \in \mathbb{N}$ and input values $\tau \in \{0,1\}^\kappa$,

$$\Pr[\mathsf{PP} \leftarrow \mathsf{VRF.Setup}(1^\kappa); (\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{VRF.Gen}(); (T, \pi_{\mathsf{vrf}}) \xleftarrow{\$} \mathsf{VRF.Prove}(\mathsf{sk}, \tau) : \mathsf{VRF.Ver}(\mathsf{vk}, \tau, T, \pi_{\mathsf{vrf}}) = 1] = 1$$

2. **Uniqueness.** For all PPT adversaries $\mathcal{A}$ playing game $\mathsf{Game}_{\mathcal{A}}^{\text{unique}}$ (Fig. 1), there exists a negligible function ν such that:

$$\mathsf{Adv}_{\mathcal{A}}^{\text{unique}}(\kappa) = \Pr\left[\mathsf{Game}_{\mathcal{A}}^{\text{unique}}(\kappa) = 1\right] \leq \nu(\kappa)$$

3. **Pseudorandomness.** For all PPT adversaries $\mathcal{A}$ playing game $\mathsf{Game}_{\mathcal{A}}^{\text{pseudo}}$ (Fig. 2), there exists a negligible function ν such that:

$$\mathsf{Adv}_{\mathcal{A}}^{\text{pseudo}}(\kappa) = \Pr\left[\mathsf{Game}_{\mathcal{A}}^{\text{pseudo}}(\kappa) = 1\right] \leq \nu(\kappa)$$

Our Asymmetric Dodis-Yampolskiy VRF Construction. Our asymmetric Dodis-Yampolskiy VRF ($\mathsf{DY\text{-}VRF}^\dagger$) construction is specified in Fig. 4.

Theorem 1. *Our asymmetric DY VRF (Fig. 4) satisfies uniqueness (Fig. 1) and pseudorandomness (Fig. 2) under the q-Decisional Bilinear Diffie-Hellman Inversion assumption (Assumption 2).*

Proof. This follows from the uniqueness and pseudorandomness of the DY VRF under the symmetric q-Decisional Bilinear Diffie-Hellman Inversion assumption (q-DBDHI) [DY05].

In the original Dodis-Yampolskiy reduction, the powers $(g, g^x, \ldots, g^{x^q})$ are used to simulate the output and VRF proof. The additional powers $(\hat{g}, \hat{g}^x, \ldots, \hat{g}^{x^q})$ in our construction are only needed to simulate the VRF proof $\hat{\pi}_{\mathsf{vrf}}$ in the other source group. We note that the security proof given in [DY05] is for small inputs, which aligns with identifiers s being small.

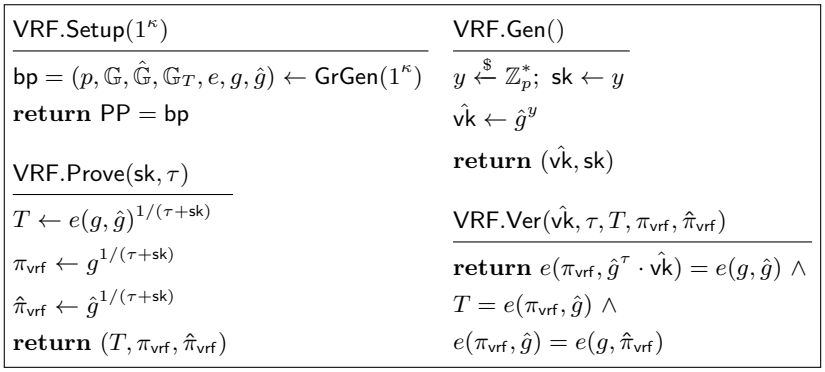

Fig. 4. Our asymmetric Dodis-Yampolskiy VRF ($\mathsf{DY\text{-}VRF}^\dagger$).

3 Formal Modelling of SyRA Signatures

We now define Sybil-resilient anonymous signatures in the form of an ideal functionality $\mathcal{F}_{\mathsf{SyRA}}$, which captures the desired security properties: unforgeability, Sybil resilience, privacy, and unlinkability.

Session identifiers are of the form $\mathsf{sid} = (\mathsf{Iss}, \mathsf{sid}')$. Initially, $\mathsf{init} \leftarrow 0$. At the end of *Key Generation*, init is set to 1. In the beginning of all other interfaces of the functionality (*Issue*, *Signature Generation*, *Verification and Extract Pseudonym*, and *Leak Signatures*) it is checked whether init has been set to 1. If not, $\mathcal{F}_{\mathsf{SyRA}}$ ignores the received message.

Key Generation. The issuer's party identifier Iss is included in the session identifier sid. $\mathcal{F}_{\mathsf{SyRA}}$ first checks that $\mathsf{sid} = (\mathsf{Iss}, \mathsf{sid}')$, which guarantees that each issuer can initialize its own instance of the functionality. $\mathcal{F}_{\mathsf{SyRA}}$ then asks the adversary $\mathcal{A}$ for a verification key ivk.

Issue. Before being given the ability to sign messages, a party has to provide evidence that they are indeed the rightful owner of the identity string s. We treat the verification of this evidence abstractly via a personhood relation R. The relation takes a public value $\mathtt{x}_s$, which specifies the public information known to both the enrolling party and the issuer, while $s, \mathtt{w}_s$ is the private input of the enrolling party.

The personhood relation allows us to configure the functionality to support different identification processes. For instance, if users are in possession of X509 attribute certificates for their s value, the relation can be of the form $R_{Gov}(\mathtt{x}_s, s, \mathtt{w}_s)$ so that $\mathtt{x}_s = (\mathsf{pk}_{Gov}, \mathsf{cert}_U), \mathtt{w}_s = \mathsf{sk}_U$, $\mathsf{cert}_U = (\mathsf{pk}_U, s)$, and $\mathsf{Cert.Verify}(\mathsf{pk}_{Gov}, \mathsf{cert}_U) = 1 \wedge \mathsf{pk}_U = g^{\mathsf{sk}_U}$. That is, the user proves that they possess a certificate on s signed by the government for which they know their secret key. Note that it is crucial that the issuer confirms pk_{Gov} as part of its input; otherwise, the user could use certificates signed by arbitrary public keys.

It is straightforward to extend the above to a setting where various identity providers are recognized by the issuer. Another alternative is that the issuer and user run DECO [ZMM+20], a mechanism for porting existing identities from websites (e.g., bank or government websites). Note also that $\mathtt{x}_s$ models the leakage of the certification process. If one were to, for example, model anonymous credentials instead of classical certificates, cert_U would become part of $\mathtt{w}_s$ and $\mathtt{x}_s$ would not reveal any information about s. We intentionally leave any further details about R and the way it is implemented in the environment unspecified, so that our mechanism is compatible with a variety of different personhood instantiations.

Signature Generation. Both honest and malicious parties can request signatures on a context ctx and a message m. Upon receiving the sign instruction, a new signature will be requested from the adversary $\mathcal{A}$. In this request, we only leak the identity of compromised users together with a value ucid, which is a unique label per each user-context pair. Recall that a user is compromised if its s value has been submitted to the issue protocol of a malicious party—modelling different devices of the user.

Upon a response from the adversary, the functionality adds a record to the set *Sig* to enable signature verification. The adversary submits the signature σ and pseudonym T to the ideal functionality together with s^* of malicious users as she can hold several identity strings (s^* is ignored for honest users). $\mathcal{F}_{\mathsf{SyRA}}$ checks if for the same ivk, s, and ctx the submitted T is consistent with already recorded T' or not. $\mathcal{F}_{\mathsf{SyRA}}$ also checks the uniqueness of T.

Verification. The verification interface allows external parties to verify signatures and extract the associated pseudonyms of the signatures. Note that we do not assume a public-key infrastructure for issuer verification keys or authenticated channels between verifiers and issuers; thus, verifiers might inadvertently use incorrect issuer verification keys.

For signatures generated with respect to the session's issuer verification key, verification is based on the records in *Sig* created during signature generation. For other verification keys, the adversary decides the verification outcome and we add new signature records to the set *Sig* to assure verification consistency.

Regarding pseudonym extraction, the interface also allows external parties to submit signatures and receive the associated pseudonym. In this way, with the pseudonym in hand, parties are able to check their local key-value store table to see whether (pseudonym, context) exists, efficiently detecting all signatures of the same s holder on the same context ctx.

If the submitted signature is invalid (for ivk), $\mathcal{F}_{\mathsf{SyRA}}$ returns $\perp$ for the pseudonym. Additionally, $\mathcal{F}_{\mathsf{SyRA}}$ guarantees consistency by adding a record of previous verification and pseudonym extraction results to the set *Ver*.

Leak Signatures. This is an adversarial interface for retrieving the signatures of users who are corrupted during the protocol (i.e., adaptive corruptions).

Functionality Sybil-Resilient Anonymous Signatures $\mathcal{F}_{\mathsf{SyRA}}$

Key Generation.

1. Upon input $(\texttt{Init}, \mathsf{sid})$ from some party P: Parse $\mathsf{sid} = (\mathsf{Iss}, \mathsf{sid}')$, ignore if $P \neq \mathsf{Iss}$. Else, output $(\texttt{Init}, \mathsf{sid}, P)$ to $\mathcal{A}$.
2. Upon receiving $(\texttt{VerKey}, \mathsf{sid}, \mathsf{ivk})$ from $\mathcal{A}$: Output $(\texttt{VerKey}, \mathsf{sid}, \mathsf{ivk})$ to P. Record ivk. Set $\mathsf{init} \leftarrow 1$.

Issue.

1. Upon receiving a value $(\texttt{Issue}, \mathsf{sid}, s, \mathtt{x}_s, \mathtt{w}_s)$ from some party P, and $(\texttt{Issue.Ok}, \mathsf{sid}, \mathsf{P}, \mathtt{x}_s)$ from party Iss, where $\mathsf{sid} = (\mathsf{Iss}, \mathsf{sid}')$: Ignore if $R(\mathtt{x}_s, (s, \mathtt{w}_s)) \neq 1$ or if P is honest and there exists s' such that $\mathbb{I}(\mathsf{P}) = s'$ and $s' \neq s$. Else, act as follows:
   (a) If P is malicious, retrieve $\{\mathsf{P}_i\}_i$ values of honest parties for which $\{\mathbb{I}(\mathsf{P}_i) = s\}_i$ holds. Set $\hat{\mathcal{P}}(s) \leftarrow \{\mathsf{P}_i\}_i$.
   (b) Else (P is honest), if there exists a malicious party P' for which $s \in \mathbb{I}(\mathsf{P}')$ holds, set $\hat{\mathcal{P}}(s) \leftarrow \hat{\mathcal{P}}(s) \cup \mathsf{P}$.
   Generate a fresh id. Set $T(\mathsf{id}) \leftarrow (\mathsf{P}, s)$. If $\mathsf{P} \in \hat{\mathcal{P}}(s)$, set $\mathcal{T} \leftarrow (\mathsf{P}, s, \hat{\mathcal{P}}(s), \mathtt{x}_s)$. Else, set $\mathcal{T} \leftarrow \mathtt{x}_s$. Hand $(\texttt{Issue}, \mathsf{sid}, \mathsf{id}, \mathcal{T})$ to $\mathcal{A}$.
2. Upon receiving $(\texttt{Issued}, \mathsf{sid}, \mathsf{id})$ from $\mathcal{A}$: Ignore if $T(\mathsf{id}) = \perp$. Else, retrieve $T(\mathsf{id}) = (\mathsf{P}, s)$. Set $\mathbb{I}(\mathsf{P}) \leftarrow \mathbb{I}(\mathsf{P}) \cup \{s\}$. Output $(\texttt{Issued}, \mathsf{sid})$ to P.

Signature Generation.

1. Upon receiving a value $(\texttt{Sign}, \mathsf{sid}, \mathsf{ctx}, m)$ from some party P: Ignore if $\mathbb{I}(\mathsf{P}) = \emptyset$. Else, if there exists a ucid where $\mathbb{M}(\mathsf{ucid}) = (\mathsf{P}, \mathsf{ctx})$, retrieve ucid. Else, generate a fresh ucid and set $\mathbb{M}(\mathsf{ucid}) \leftarrow (\mathsf{P}, \mathsf{ctx})$. If there exists an s value such that $\mathsf{P} \in \hat{\mathcal{P}}(s)$, set $\psi \leftarrow (\mathsf{ctx}, m, \mathsf{P})$. Else, set $\psi \leftarrow (\mathsf{ctx}, m)$. Select a fresh tid and set $\mathbb{D}(\mathsf{tid}) \leftarrow (\mathsf{P}, \mathsf{ctx}, m)$. Hand $(\texttt{Sign}, \mathsf{sid}, \mathsf{ucid}, \psi, \mathsf{tid})$ to $\mathcal{A}$.
2. Upon receiving $(\texttt{Signature}, \mathsf{sid}, \sigma, T, s^*, \mathsf{tid})$ from $\mathcal{A}$: Ignore if $\mathbb{D}(\mathsf{tid}) = \perp$, or there exists an entry $(\cdot, \cdot, \cdot, \cdot, \sigma, \cdot, \cdot)$ in the set Sig. Else, retrieve $\mathbb{D}(\mathsf{tid}) = (\mathsf{P}, \mathsf{ctx}, m)$.
   (a) If $s^* \in \mathbb{I}(\mathsf{P})$ and P is malicious, set $s \leftarrow s^*$.
   (b) Else, if P is honest, retrieve $\mathbb{I}(\mathsf{P}) = \{s'\}$ and set $s \leftarrow s'$.
   (c) Else, ignore.
   Ignore if:
   – there exists $(\mathsf{ivk}, s, \mathsf{ctx}, \cdot, \cdot, T', \cdot) \in Sig$ where $T' \neq T$, or
   – there exists $(\cdot, \cdot, \cdot, \cdot, \cdot, T', \cdot) \in Sig$ where $T' = T$.
   Else, add $(\mathsf{ivk}, s, \mathsf{ctx}, m, \sigma, T, 1)$ to the set Sig. Output $(\texttt{Signature}, \mathsf{sid}, \mathsf{ctx}, m, \sigma)$ to P.

Verification.

1. Upon receiving a value $(\texttt{Ver}, \mathsf{sid}, \mathsf{ivk}', \mathsf{ctx}, m, \sigma)$ from some party V: Hand $(\texttt{Ver}, \mathsf{sid}, \mathsf{ivk}', \mathsf{ctx}, m, \sigma)$ to $\mathcal{A}$.
2. Upon receiving $(\texttt{Ver.Ok}, \mathsf{sid}, \mathsf{ivk}', \mathsf{ctx}, m, \sigma, T, \theta)$ from $\mathcal{A}$:
   (a) If $(\mathsf{ivk}', \cdot, \mathsf{ctx}, m, \sigma, T', \Theta') \in Ver$, set $\mathcal{T} \leftarrow T'$ and $\Theta \leftarrow \Theta'$.
   (b) Else, if $(\mathsf{ivk}, s, \mathsf{ctx}, m, \sigma, T', 1) \in Sig$ such that $\mathsf{ivk}' = \mathsf{ivk}$, record $(\mathsf{ivk}, s, \mathsf{ctx}, m, \sigma, T', 1)$ in the set Ver and set $\mathcal{T} \leftarrow T'$ and $\Theta \leftarrow 1$.

(c) Else, if $(\mathsf{ivk}, \cdot, \mathsf{ctx}, m, \sigma, \cdot, 1) \notin$ *Sig* such that $\mathsf{ivk}' = \mathsf{ivk}$, record $(\mathsf{ivk}, \cdot, \mathsf{ctx}, m, \sigma, \perp, 0)$ in the set *Ver* and set $\mathcal{T} \leftarrow \perp$ and $\Theta \leftarrow 0$.
(d) Else, record $(\mathsf{ivk}', \cdot, \mathsf{ctx}, m, \sigma, T, \theta)$ in the set *Ver* and set $\mathcal{T} \leftarrow T$ and $\Theta \leftarrow \theta$.

Output $(\texttt{Ver.End}, \mathsf{sid}, \mathsf{ivk}', \mathsf{ctx}, m, \sigma, \mathcal{T}, \Theta)$ to V.

Leak Signatures.

1. Upon receiving $(\texttt{Leak.Sig}, \mathsf{sid}, \mathsf{ivk}, s)$ from $\mathcal{A}$: If $\hat{\mathcal{P}}(s) = \emptyset$, set $\eta \leftarrow \perp$. Else, retrieve $\{(\mathsf{ivk}, s, \cdot, \cdot, \sigma_i, T_i, 1)\}_i$ from set *Sig* and set $\eta \leftarrow \{\sigma_i\}_i$. Hand $(\texttt{Leaked.Sig}, \mathsf{sid}, \mathsf{ivk}, \eta)$ to $\mathcal{A}$.

4 Our SyRA Signature Construction

We describe our Sybil-resilient anonymous signature construction Π_{SyRA} and prove that Π_{SyRA} securely realizes $\mathcal{F}_{\mathsf{SyRA}}$. Our construction uses two cryptographic primitives: ElGamal encryption (Definition 3) and our asymmetric Dodis-Yampolskiy verifiable random function DY-VRF† (Section 2.2). Additionally, it relies on the following ideal sub-functionalities: random oracle functionality $\mathcal{F}_{\mathsf{RO}}$ (Definition 5), non-interactive zero knowledge functionality $\mathcal{F}_{\mathsf{NIZK}}$ (Definition 6), and (communication) channel functionality $\mathcal{F}_{\mathsf{Ch}}$ (Definition 7), parameterized by two labels: (i) SSA for *secure and sender anonymous* channel $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SSA}}$, (ii) SRA for *secure and receiver anonymous* channel $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SRA}}$.[6] The construction Π_{SyRA} is presented in four phases: *Key Generation, Issue, Signature Generation*, and *Verification* as follows.

Key Generation. In this phase, the issuer Iss is activated by $\mathcal{Z}$, and generates its secret signing key and public verification key pair $(\mathsf{isk}, \mathsf{ivk})$.
Iss acts as follows upon receiving $(\texttt{Init}, \mathsf{sid})$ from $\mathcal{Z}$:

1. Call $\mathsf{GrGen} : \mathsf{bp} = (p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g, \hat{g}) \leftarrow \mathsf{GrGen}(1^k)$, where g and $\hat{g}$ are generators of $\mathbb{G}$ and $\hat{\mathbb{G}}$, respectively.
2. Pick $\mathsf{isk} \xleftarrow{\$} \mathbb{Z}_p^*, W \xleftarrow{\$} \mathbb{G}$, and $\hat{W} \xleftarrow{\$} \hat{\mathbb{G}}$.
3. Set $\hat{\mathsf{ivk}} \leftarrow \hat{g}^{\mathsf{isk}}$ and $\mathsf{ivk} \leftarrow (\mathsf{bp}, \hat{\mathsf{ivk}}, W, \hat{W})$.
4. Output $(\texttt{VerKey}, \mathsf{sid}, \mathsf{ivk})$ to $\mathcal{Z}$.

Issue. Issue is an interactive protocol between the party P offering a unique real-world identifier s and the issuer Iss holding isk. P is activated by $\mathcal{Z}$. The

[6] Note that a degree of sender/receiver anonymity is essential for maintaining privacy. Without this anonymity, "network leakage" could potentially expose the parties involved in a transaction, regardless of the robustness of cryptographic safeguards at the transactional level. Furthermore, in real-world deployment, some level of network leakage might be deemed acceptable. In such a scenario, our analysis would still be applicable, acknowledging that the adversary could compromise privacy through traffic analysis to some extent.

protocol produces the secret key $(\mathsf{usk}, \hat{\mathsf{usk}})$ of a party, which corresponds to the two VRF proofs, π_{vrf} and $\hat{\pi}_{\mathsf{vrf}}$, of DY-VRF†. There is no output for Iss.

P acts as follows upon receiving $(\mathtt{Issue}, \mathsf{sid}, s, \mathtt{x}_s, \mathtt{w}_s)$ from $\mathcal{Z}$:

1. Ignore, if there exists a recorded s' such that $s' \neq s$.
2. Else, call $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Prove}, \mathsf{sid}, \mathtt{x}_s, (s, \mathtt{w}_s))$ for the relation $R(\mathtt{x}_s, (s, \mathtt{w}_s))$.
3. Upon receiving $(\mathtt{Proof}, \mathsf{sid}, \pi_s)$ from $\mathcal{F}_{\mathsf{NIZK}}$, call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SSA}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{Iss}, (s, \mathtt{x}_s, \pi_s))$.
4. Record s as s'.

Iss, upon receiving $(\mathtt{Received}, \mathsf{sid}, \mathsf{P}, (s, \mathtt{x}'_s, \pi_s))$ and $(\mathtt{Issue.Ok}, \mathsf{sid}, \mathsf{P}, \mathtt{x}_s)$, where $\mathtt{x}'_s = \mathtt{x}_s$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SSA}}$ and $\mathcal{Z}$, respectively, acts as follows:

1. Call $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}_s, \pi_s)$ and receive $(\mathtt{Verification}, \mathsf{sid}, b)$.
2. Ignore if $b = 0$. Else, compute the user secret key: $(\mathsf{usk}, \hat{\mathsf{usk}}) \leftarrow (g^{1/(s+\mathsf{isk})}, \hat{g}^{1/(s+\mathsf{isk})})$.
3. Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SRA}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{P}, (\mathsf{ivk}, \mathsf{usk}, \hat{\mathsf{usk}}))$.

Upon receiving $(\mathtt{Received}, \mathsf{sid}, \mathsf{Iss}, (\mathsf{ivk}, \mathsf{usk}, \hat{\mathsf{usk}}))$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SRA}}$, P records $(\mathsf{ivk}, \mathsf{usk}, \hat{\mathsf{usk}})$ and outputs $(\mathtt{Issued}, \mathsf{sid})$ to $\mathcal{Z}$.

Signature Generation. To sign messages, party P is activated by $\mathcal{Z}$ with a message $(\mathtt{Sign}, \mathsf{sid}, \mathsf{ctx}, m)$. P, who has an identity string s, a secret key $(\mathsf{usk}, \hat{\mathsf{usk}})$, a context-message pair (ctx, m), and issuer verification key ivk generates a signature σ on (ctx, m) as follows:

1. Ignore if $(\mathsf{ivk}, \mathsf{usk}, \hat{\mathsf{usk}})$ has not been recorded. Else, proceed as follows.
2. Call $\mathcal{F}_{\mathsf{RO}}$ with $(\mathtt{Query}, \mathsf{sid}, \mathsf{ctx})$ and receive $(\mathtt{Query.Re}, \mathsf{sid}, Z)$. Compute $T \leftarrow e(Z, \hat{\mathsf{usk}})$.
3. Pick $(\beta, \alpha) \xleftarrow{\$} \mathbb{Z}_p^*$. Parse $\mathsf{ivk} = (\mathsf{bp}, \hat{\mathsf{ivk}}, W, \hat{W})$. Compute $C = (C_1, C_2) \leftarrow (g^{\beta}, W^{\beta} \cdot \mathsf{usk})$ and $\hat{C} = (\hat{C}_1, \hat{C}_2) \leftarrow (\hat{g}^{\alpha}, \hat{W}^{\alpha} \cdot \hat{\mathsf{usk}})$.
4. Call $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Prove}, \mathsf{sid}, \mathtt{x}, \mathtt{w})$ for the following relation $R(\mathtt{x}, \mathtt{w})$, statement $\mathtt{x}$, and witness $\mathtt{w}$:
   - $R(\mathtt{x}, \mathtt{w}) \Leftrightarrow e(Z, \hat{W})^{\alpha} = e(Z, \hat{C}_2)/T \wedge e(C_2, \hat{g})e(g^{-1}, \hat{C}_2) = e(W, \hat{g})^{\beta} e(g^{-1}, \hat{W})^{\alpha} \wedge e(C_2, \hat{\mathsf{ivk}})e(g^{-1}, \hat{g}) = e(W, \hat{\mathsf{ivk}})^{\beta} e(W, \hat{g})^{\beta \cdot s} e(C_2, \hat{g}^{-1})^{s} \wedge C_1 = g^{\beta} \wedge \hat{C}_1 = \hat{g}^{\alpha}$.
   - $\mathtt{x} = (Z, C, \hat{C}, T, \mathsf{ivk}, \mathsf{ctx}, m)$
   - $\mathtt{w} = (s, \alpha, \beta)$
5. Upon receiving $(\mathtt{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{F}_{\mathsf{NIZK}}$, output $(\mathtt{Signature}, \mathsf{sid}, \mathsf{ctx}, m, \sigma = (\pi, \mathtt{x}))$ to $\mathcal{Z}$.

The signer must prove the well-formedness of the pseudonym $T = e(Z, \hat{\mathsf{usk}})$ without revealing their secret key $\hat{\mathsf{usk}}$. This is accomplished by encrypting $\hat{\mathsf{usk}}$ under a public group element $\hat{W}$. Hence, instead of proving $T = e(Z, \hat{\mathsf{usk}})$, they prove $e(Z, \hat{W})^{\alpha} = e(Z, \hat{C}_2)/T$. Additionally, the signer needs to prove that the encrypted $\hat{\mathsf{usk}}$ is well-formed with respect to the issuer verification key ivk. To

do so, the signer first proves that $\hat{\mathsf{usk}}$ is well-formed with respect to usk (by $e(C_2,\hat{g})e(g^{-1},\hat{C}_2)=e(W,\hat{g})^{\beta}e(g^{-1},\hat{W})^{\alpha}$, where usk is encrypted under a public group element W) and then proves the well-formedness of usk with respect to ivk (by $e(C_2,\hat{\mathsf{ivk}})e(g^{-1},\hat{g}) = e(W,\hat{\mathsf{ivk}})^{\beta}e(W,\hat{g})^{\beta\cdot s}e(C_2,\hat{g}^{-1})^{s}$). Finally, the well-formedness of C_1 and $\hat{C}_1$ is proven.

Verification. The verifier V is activated upon receiving a message $(\mathtt{Ver},\mathsf{sid},\mathsf{ivk}',\mathsf{ctx},m,\sigma)$ from $\mathcal{Z}$ that includes the issuer verification key ivk', the context-message pair (ctx,m), and the signature σ. The signature is verified and the associated pseudonym is extracted as follows:

1. Parse $\sigma=(\pi,\mathtt{x})$ and $\mathtt{x}=(Z,C,\hat{C},T,\mathsf{ivk},\mathsf{ctx},m)$.
2. Set $b\leftarrow 0$ and $\eta\leftarrow\perp$ if:
   - $\mathsf{ivk}\neq\mathsf{ivk}'$; or
   - upon calling $\mathcal{F}_{\mathsf{RO}}$ with $(\mathtt{Query},\mathsf{sid},\mathsf{ctx})$, $(\mathtt{Query.Re},\mathsf{sid},Z')$ is returned where $Z'\neq Z$; or
   - upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify},\mathsf{sid},\mathtt{x},\pi)$, $(\mathtt{Verification},\mathsf{sid},0)$ is returned.
3. Else, set $b\leftarrow 1$ and $\eta\leftarrow T$.
4. Output $(\mathtt{Ver.End},\mathsf{sid},\mathsf{ivk}',\mathsf{ctx},m,\sigma,\eta,b)$ to $\mathcal{Z}$.

4.1 Implementing Our Non-Interactive Zero-Knowledge Proofs

In our signature framework, the relation $R(\mathtt{x},\mathtt{w})$ that is proven by the signer is expressed as a set of algebraic discrete logarithm equations. Sigma protocols excel in prover efficiency when applied to such algebraic statements compared to zk-SNARKs [Gro10]. Sigma protocols result in concise proof sizes, involve a limited number of operations, and do not necessitate the creation of a trusted common reference string [GQ88,Sch91]. This observation is particularly relevant in our system, where proof generation for algebraic statements could mainly depend on individuals with limited computational resources.

In the following, the prover and the verifier are denoted by Prv and Vrf, respectively. Prv proves the following relation:

- $R(\mathtt{x},\mathtt{w})\Leftrightarrow e(Z,\hat{W})^{\alpha}=e(Z,\hat{C}_2)/T\wedge e(C_2,\hat{g})e(g^{-1},\hat{C}_2)=e(W,\hat{g})^{\beta}e(g^{-1},\hat{W})^{\alpha}$ $\wedge\, e(C_2,\hat{\mathsf{ivk}})e(g^{-1},\hat{g})=e(W,\hat{\mathsf{ivk}})^{\beta}e(W,\hat{g})^{\beta\cdot s}e(C_2,\hat{g}^{-1})^{s}\wedge C_1=g^{\beta}\wedge\hat{C}_1=\hat{g}^{\alpha}$.
- $\mathtt{x}=(Z,C,\hat{C},T,\mathsf{ivk},\mathsf{ctx},m)$
- $\mathtt{w}=(s,\alpha,\beta)$

Let us define $A=e(Z,\hat{W}), B=e(Z,\hat{C}_2)/T, E=e(C_2,\hat{g})e(g^{-1},\hat{C}_2), F=e(W,\hat{g})$, $G=e(g^{-1},\hat{W}), H=e(C_2,\hat{\mathsf{ivk}})e(g^{-1},\hat{g}), I=e(W,\hat{\mathsf{ivk}})$, and $J=e(C_2,\hat{g}^{-1})$. Hence, the relation is simplified to the following one:

- $R(\mathtt{x}, \mathtt{w}) \Leftrightarrow A^{\alpha} = B \wedge \hat{C}_1 = \hat{g}^{\alpha} \wedge C_1 = g^{\beta} \wedge E = F^{\beta} G^{\alpha} \wedge H = I^{\beta} F^{\beta \cdot s} J^{s}$
- $\mathtt{x} = (Z, C, \hat{C}, T, \mathsf{ivk}, \mathsf{ctx}, m)$
- $\mathtt{w} = (s, \alpha, \beta)$

where bases (A, B, E, F, G, H, I, J) are target group elements that can be efficiently computed by the verifier Vrf using the statement $\mathtt{x} = (Z, C, \hat{C}, T, \mathsf{ivk})$. Moreover, all source group elements $(C_1, \hat{C}_1, g, \hat{g})$ are known to Vrf.

The following interactive proofs can be made non-interactive using the Fiat-Shamir transformation [FS87].[7]

Proof of Knowledge of Discrete Log:

- $R(\mathtt{x}, \mathtt{w}) \Leftrightarrow y = g^{x}$
- $\mathtt{x} = (y, g)$
- $\mathtt{w} = x$

1. Prv computes $a \leftarrow g^{\theta}$ for $\theta \xleftarrow{\$} \mathbb{Z}_q^*$. Sends a to Vrf.
2. Vrf selects $c \xleftarrow{\$} \mathbb{Z}_q$. Sends c to Prv.
3. Prv computes $z = \theta + cx \bmod q$. Sends z to Vrf.
4. Vrf checks if $a = g^{z} y^{-c}$ holds. If so, the verifier accepts.

Proof of Multiplicative Relation for Exponents:

- $R(\mathtt{x}, \mathtt{w}) \Leftrightarrow A_1 = g^{a_1} h^{r_1}, A_2 = g^{a_2} h^{r_2}, A_3 = g^{a_3} h^{r_3} = g^{a_1 a_2} h^{r_3}$
- $\mathtt{x} = (g, h, A_1, A_2, A_3)$
- $\mathtt{w} = (a_1, a_2, a_3, r_1, r_2, r_3)$
- We have to also prove that $A_3 = A_1^{a_2} h^{r}$, where $r + r_1 a_2 = r_3 \bmod q$.

1. Prv computes $v_i = g^{\theta_i} h^{R_i}$ for $i = 1, 2, 3, v = A_1^{\theta_2} h^{R}$ for $(\theta_i, R_i, \theta, R) \xleftarrow{\$} \mathbb{Z}_q$. Sends $(\{v_i\}, v)$ for $i = 1, 2, 3$ to Vrf.
2. Vrf chooses a challenge $c \xleftarrow{\$} \mathbb{Z}_{2^k}$ (k is fixed where $2^k < q$). Sends c to Prv.
3. Prv computes $s_i = \theta_i - c a_i, t_i = R_i - c r_i, t = R - c r$. Sends the tuple (s_i, t_i) for $i = 1, 2, 3$ and t to Vrf.
4. Vrf Checks if $v_i = (A_i)^{c} g^{s_i} h^{t_i}$ for $i = 1, 2, 3$ and $v = A_3^{c} A_1^{s_2} h^{t}$ hold. Vrf accepts if all four equations hold.

5 Security Proof for Our SyRA Construction

In this section, we prove the security of our Sybil-resilient anonymous signature scheme (Section 4). Our functionality, defined in Section 3, is general and allows for adaptive corruptions; however, we prove our scheme secure in the static corruption model. Concretely, this corresponds to $\cup_s \hat{\mathcal{P}}(s) = \emptyset$ in our functionality, and no Leak Signatures interface.

[7] The context-message pair (ctx, m), which is part of the NIZK statement, is included in the Fiat-Shamir hash function computation by both the signer and the verifier. This has been abstracted away by $\mathcal{F}_{\mathsf{NIZK}}$.

Theorem 2. *Let q_t be an upper bound on the number of signatures of all honest parties and let q_h be an upper bound on the number of queries the adversary can make to the random oracle. Under the* IND-CPA *security of ElGamal encryption, the pseudorandomness of our asymmetric Dodis-Yampolsky VRF (*DY-VRF†*), and the* DDH *assumption, in the $\{\mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{Ch}}\}$-hybrid model, no PPT environment $\mathcal{Z}$ with static corruptions can distinguish the real-world execution $\mathsf{EXEC}_{\Pi_{\mathsf{SyRA}},\mathcal{A},\mathcal{Z}}$ from the ideal-world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{SyRA}},\mathcal{S},\mathcal{Z}}$ with advantage greater than*

$$2q_t \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}}(k) + (q_t + 1) \cdot \mathsf{Adv}^{pseudo}_{\mathcal{A},\mathsf{DY\text{-}VRF}^\dagger}(k) + q_t \cdot q_h \cdot \mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{A}}(k)$$

in the presence of an arbitrary number of corrupted signers and verifiers that are all colluding.

5.1 Our UC Simulator

We refer to the real-world protocol as Π_{SyRA} and the adversary as $\mathcal{A}$. The simulator $\mathcal{S}$ is described in detail below. It ensures that the view of the real-world execution $\mathsf{EXEC}_{\Pi_{\mathsf{SyRA}},\mathcal{A},\mathcal{Z}}$ and the ideal-world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{SyRA}},\mathcal{S},\mathcal{Z}}$ are indistinguishable for any PPT environment $\mathcal{Z}$. The session identifier sid is selected by the environment $\mathcal{Z}$. Our general proof strategy is for the simulator $\mathcal{S}$ to internally run an instance of Π_{SyRA} and the real-world adversary $\mathcal{A}$, except that $\mathcal{S}$ simulates the behaviour of honest users. The purpose of the simulator $\mathcal{S}$ is to ensure that the view of the internally-run adversary $\mathcal{A}$ (in the ideal world) cannot be distinguished from the view of the real-world adversary $\mathcal{A}$. This enables $\mathcal{S}$ to extract the necessary information, so that it can instruct the functionality to also provide indistinguishable outputs at the interfaces of honest users.

At the beginning of the execution, the environment $\mathcal{Z}$ instructs the adversary to corrupt certain parties using a message of the form $(\texttt{Corrupt}, \mathsf{sid}, \mathsf{P})$, where P represents a party that can be any entity within the network, excluding the issuer Iss. The simulator $\mathcal{S}$ reads these corruption messages and informs $\mathcal{F}_{\mathsf{SyRA}}$ about which parties have been corrupted by sending the message $(\texttt{Corrupt}, \mathsf{sid}, \mathsf{P})$. Additionally, the simulator $\mathcal{S}$ keeps track of the identifiers of the corrupted parties for future reference. The adversary $\mathcal{A}$ has the authority to instruct corrupted parties in an arbitrary manner, and the simulator $\mathcal{S}$ engages in interactions with $\mathcal{F}_{\mathsf{SyRA}}$ on behalf of these corrupted parties. In the ideal-world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{SyRA}},\mathcal{S},\mathcal{Z}}$, the honest (dummy) parties transmit their inputs from the environment $\mathcal{Z}$ directly to $\mathcal{F}_{\mathsf{SyRA}}$.

We define a simulator $\mathcal{S}$ that reproduces the real-world view of the adversary $\mathcal{A}$ and simulates the actions of honest parties during the execution. The simulator $\mathcal{S}$ engages in interactions with both the dummy adversary $\mathcal{A}$ and the functionality $\mathcal{F}_{\mathsf{SyRA}}$. Similar to the functionality $\mathcal{F}_{\mathsf{SyRA}}$ and our protocol Π_{SyRA} for realizing it, the simulator's details are described in four parts: *Key Generation, Issue, Signature Generation*, and *Verification*. As $\cup_s \hat{\mathcal{P}}(s) = \emptyset$, we do not consider the *Leak Signatures* interface ($(\texttt{Leaked.Sig}, \mathsf{sid}, \mathsf{ivk}, \perp)$ is always sent to the simulator). The simulator $\mathcal{S}$ emulates the functionalities $\mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{RO}}$, and

$\mathcal{F}_{\mathsf{Ch}}$. To achieve this, it must maintain specific lists associated with each functionality. However, for the sake of simplicity and without loss of generality, we assume that $\mathcal{S}$ keeps track of the states of these functionalities, but we do not explicitly discuss or address all these lists in detail.

The simulator $\mathcal{S}$ maintains the following lists and sets, which are initialized to $\emptyset$:

- $\mathcal{L}_{\mathsf{CR}}$: list of corrupted-registered parties (who have been issued signing keys)
- $\mathbb{S}^{\mathsf{corrupted}}$: set of s values of corrupted parties
- $\mathcal{L}_{\mathsf{SS}}$: list of simulated signatures (by the simulator $\mathcal{S}$)

Key Generation

1. Honest issuer Iss:
   (a) Receive $(\mathtt{Init}, \mathsf{sid}, P)$ from $\mathcal{F}_{\mathsf{SyRA}}$ and start simulating the honest issuer (similar to the real-world scheme).
   (b) Execute the GrGen algorithm and generate the system's public parameters ivk following the real-world algorithm.
   (c) Submit $(\mathtt{VerKey}, \mathsf{sid}, \mathsf{ivk})$ to $\mathcal{F}_{\mathsf{SyRA}}$.

Issue

1. Honest party P and honest issuer Iss:
   (a) Upon receiving $(\mathtt{Issue}, \mathsf{sid}, \mathsf{id}, \mathtt{x}_s)$ from $\mathcal{F}_{\mathsf{SyRA}}$, start emulating honest P.
   (b) Leak $(\mathtt{Prove}, \mathsf{sid}, \mathtt{x}_s)$ to $\mathcal{A}$ (as the leakage the real-world $\mathcal{A}$ sees from $\mathcal{F}_{\mathsf{NIZK}}$ once the honest P calls $\mathcal{F}_{\mathsf{NIZK}}$).
   (c) Emulating $\mathcal{F}_{\mathsf{NIZK}}$, store $(\mathtt{x}_s, \pi_s)$ upon receiving $(\mathtt{Proof}, \mathsf{sid}, \pi_s)$ from $\mathcal{A}$.
   (d) Leak $(\mathtt{Send}, \mathsf{sid}, \mathsf{Iss}, |m|, \mathsf{mid})$ to the dummy $\mathcal{A}$ (which is what the real-world $\mathcal{A}$ sees as the leakage of the communication channel $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SSA}}$ between P and Iss), where the message m is of the form $(s, \mathtt{x}_s, \pi_s)$ whose size $|m|$ is known to $\mathcal{S}$.
   (e) Upon receiving $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid})$ from $\mathcal{A}$, leak $(\mathtt{Send}, \mathsf{sid}, \mathsf{Iss}, |m|, \mathsf{mid}')$ to $\mathcal{A}$ (which is what the real-world $\mathcal{A}$ sees as the leakage of the communication channel $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SRA}}$ between Iss and P), where m is of the form $(\mathsf{ivk}, \mathsf{usk}, \hat{\mathsf{usk}})$ whose size $|m|$ is known to $\mathcal{S}$.
   (f) Upon receiving $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid}')$ from $\mathcal{A}$, submit $(\mathtt{Issued}, \mathsf{sid}, \mathsf{id})$ to $\mathcal{F}_{\mathsf{SyRA}}$.
2. Corrupted party P and honest issuer Iss:
   (a) Once $\mathcal{A}$ (on behalf of a corrupted party P) calls $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SSA}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{Iss}, (s, \mathtt{x}_s, \pi_s))$,[8] leak $(\mathtt{Send}, \mathsf{sid}, \mathsf{Iss}, |m|, \mathsf{mid})$ to $\mathcal{A}$ (which is what the real-world $\mathcal{A}$ sees as the leakage of the communication channel $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SSA}}$ between P and Iss), where $m = (s, \mathtt{x}_s, \pi_s)$ is received via $\mathcal{S}$ who emulates $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SSA}}$.
   (b) Provide $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}_s, \pi_s)$ to $\mathcal{A}$.

[8] For an honest Iss to receive $(\mathtt{Received}, \mathsf{sid}, \mathsf{P}, (s, \mathtt{x}_s, \pi_s))$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SSA}}$, $\mathcal{A}$ should call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SSA}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{Iss}, (s, \mathtt{x}_s, \pi_s))$. Otherwise, honest Iss does not proceed.

(c) Upon receiving $(\texttt{Witness}, \mathsf{sid}, \texttt{w}'_s)$ from $\mathcal{A}$ where $\texttt{w}'_s = (s, \texttt{w}_s)$, submit $(\texttt{Issue}, \mathsf{sid}, s, \texttt{x}_s, \texttt{w}_s)$ on behalf of P to $\mathcal{F}_{\mathsf{SyRA}}$.[9]

(d) Upon receiving $(\texttt{Issue}, \mathsf{sid}, \mathsf{id}, \texttt{x}_s)$ from $\mathcal{F}_{\mathsf{SyRA}}$ (which means $R(\texttt{x}_s, (s, \texttt{w}_s))$ has already been checked by $\mathcal{F}_{\mathsf{SyRA}}$), compute $\mathsf{usk} = g^{1/(s+\mathsf{isk})}$ and $\hat{\mathsf{usk}} = \hat{g}^{1/(s+\mathsf{isk})}$.

(e) Leak $(\texttt{Send}, \mathsf{sid}, \mathsf{lss}, |m|, \mathsf{mid}')$, where $m = (\mathsf{ivk}, \mathsf{usk}, \hat{\mathsf{usk}})$, to $\mathcal{A}$ (which is what the real-world $\mathcal{A}$ sees as the leakage of the communication channel $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SRA}}$ between lss and P).

(f) Upon receiving $(\texttt{Ok}, \mathsf{sid}, \mathsf{mid}')$ from $\mathcal{A}$, provide $(\texttt{Received}, \mathsf{sid}, \mathsf{lss}, (\mathsf{ivk}, \mathsf{usk}, \hat{\mathsf{usk}}))$ to $\mathcal{A}$ emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SRA}}$.

(g) Set $\mathcal{L}_{\mathsf{CR}} \leftarrow \mathcal{L}_{\mathsf{CR}} \cup \{\mathsf{P}\}$.

(h) Set $\mathbb{S}^{\mathsf{corrupted}} \leftarrow \mathbb{S}^{\mathsf{corrupted}} \cup \{s\}$.

(i) Submit $(\texttt{Issued}, \mathsf{sid}, \mathsf{id})$ to $\mathcal{F}_{\mathsf{SyRA}}$ (and output to $\mathcal{Z}$ whatever $\mathcal{A}$ outputs).

Signature Generation

1. Honest party P:
   (a) Receive $(\texttt{Sign}, \mathsf{sid}, \mathsf{ucid}, \mathsf{ctx}, m, \mathsf{tid})$ from $\mathcal{F}_{\mathsf{SyRA}}$.
   (b) Emulating $\mathcal{F}_{\mathsf{RO}}$, retrieve Z.
   (c) If there exists an entry (ucid', T') recorded where $\mathsf{ucid}' = \mathsf{ucid}$, set $T \leftarrow T'$. Else, pick $T \xleftarrow{\$} \mathbb{G}_T$, and record (ucid, T).
   (d) Pick $(\beta, \alpha) \xleftarrow{\$} \mathbb{Z}_p^*$ and $A \xleftarrow{\$} \mathbb{G}, \hat{B} \xleftarrow{\$} \hat{\mathbb{G}}$.
   (e) Compute $C \leftarrow (g^\beta, W^\beta \cdot A)$, and $\hat{C} \leftarrow (\hat{g}^\alpha, \hat{W}^\alpha \cdot \hat{B})$.
   (f) Leak $(\texttt{Prove}, \mathsf{sid}, \texttt{x})$ to the dummy $\mathcal{A}$, where $\texttt{x} = (Z, C, \hat{C}, T, \mathsf{ivk}, \mathsf{ctx}, m)$.
   (g) Upon receiving $(\texttt{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{A}$, emulate $\mathcal{F}_{\mathsf{NIZK}}$ record $(\pi, \texttt{x})$.
   (h) Set $\sigma \leftarrow (\pi, \texttt{x})$ and $\mathcal{L}_{\mathsf{SS}} \leftarrow \mathcal{L}_{\mathsf{SS}} \cup \{\sigma\}$.
   (i) Submit $(\texttt{Signature}, \mathsf{sid}, \sigma, T, s^*, \mathsf{tid})$ to $\mathcal{F}_{\mathsf{SyRA}}$ where $s^* = 0$.
2. Corrupted party P:
   Output to the environment $\mathcal{Z}$ whatever internally-run $\mathcal{A}$ outputs. Note that submitting a signature generation instruction to $\mathcal{F}_{\mathsf{SyRA}}$ – on behalf of corrupted P via $\mathcal{S}$ – happens later when $\mathcal{S}$, who emulates an honest verifier V, receives a (valid) signature generated by a corrupted party. It is important to note that a corrupted party can locally compute signatures. However, as long as no honest verifier has encountered any of these signatures, there is no need for any security guarantees to come into play. In other words, as we will see, once $\mathcal{Z}$ submits a valid signature for verification, the simulator first checks whether or not the signature has already been simulated by the simulator. If it is not simulated (and it is a valid signature), the simulator should register the signature to the ideal functionality $\mathcal{F}_{\mathsf{SyRA}}$ (so that it is recorded in the set Sig by $\mathcal{F}_{\mathsf{SyRA}}$). However, as long as $\mathcal{Z}$ has not submitted the signature, there is no need to do anything.

[9] $\mathcal{S}$ itself can also check whether or not $R(\texttt{x}_s, (s, \texttt{w}_s))$ holds. Moreover, the identifier of the party, P, is known to $\mathcal{S}$ who emulates the channel functionality.

Verification

1. Honest verifier V:
   (a) Receive $(\mathtt{Ver}, \mathsf{sid}, \mathsf{ivk}', \mathsf{ctx}, m, \sigma)$ from $\mathcal{F}_{\mathsf{SyRA}}$. If there exists an entry $(\mathtt{Ver.Ok}, \mathsf{sid}, \mathsf{ivk}', \mathsf{ctx}, m, \sigma, T, \theta)$, submit it to $\mathcal{F}_{\mathsf{SyRA}}$.
   (b) Else, submit $(\mathtt{Ver.Ok}, \mathsf{sid}, \mathsf{ivk}', \mathsf{ctx}, m, \sigma, \perp, 0)$ to $\mathcal{F}_{\mathsf{SyRA}}$ and record it if $\mathsf{ivk}' \neq \mathsf{ivk}$. Else, act as follows:
   (c) If there exists $\sigma' \in \mathcal{L}_{\mathsf{SS}}$ where $\sigma' = \sigma$, parse $\sigma = (\pi, \mathsf{x})$ and $\mathtt{x} = (Z, C, \hat{C}, T, \mathsf{ivk}, \mathsf{ctx}, m)$. Submit $(\mathtt{Ver.Ok}, \mathsf{sid}, \mathsf{ivk}', \mathsf{ctx}, m, \sigma, T, 1)$ to $\mathcal{F}_{\mathsf{SyRA}}$ and record it.
   (d) Else ($\sigma \notin \mathcal{L}_{\mathsf{SS}}$), act as follows:
      i. Parse $\sigma = (\pi, \mathsf{x})$ and $\mathtt{x} = (Z, C, \hat{C}, T, \mathsf{ivk}, \mathsf{ctx}, m)$. If in emulating $\mathcal{F}_{\mathsf{RO}}$ the recorded entry for context ctx is $(\mathsf{sid}, \mathsf{ctx}, h)$ where $h \neq Z$, submit $(\mathtt{Ver.Ok}, \mathsf{sid}, \mathsf{ivk}', \mathsf{ctx}, m, \sigma, \perp, 0)$ to $\mathcal{F}_{\mathsf{SyRA}}$ and record it.
      ii. Else, hand $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$ to $\mathcal{A}$.
      iii. Upon receiving $(\mathtt{Witness}, \mathsf{sid}, \mathtt{w})$ from $\mathcal{A}$, check $(\mathtt{x}, \mathtt{w}) \in R$ and if the relation does not hold, submit $(\mathtt{Ver.Ok}, \mathsf{sid}, \mathsf{ivk}', \mathsf{ctx}, m, \sigma, \perp, 0)$ to $\mathcal{F}_{\mathsf{SyRA}}$ and record it.
      iv. Else, parse $\mathtt{w} = (s, \alpha, \beta)$.
         – If $s \in \mathbb{S}^{\mathsf{corrupted}}$, submit $(\mathtt{Sign}, \mathsf{sid}, \mathsf{ctx}, m)$ to $\mathcal{F}_{\mathsf{SyRA}}$ on behalf of P where $\mathsf{P} \in \mathcal{L}_{\mathsf{CR}}$ holds. (If there exist multiple such P values, pick one at random.)
         Upon receiving $(\mathtt{Sign}, \mathsf{sid}, \mathsf{ucid}, \mathsf{ctx}, m, \mathsf{tid})$ from $\mathcal{F}_{\mathsf{SyRA}}$, parse $\sigma = (\pi, \mathsf{x})$ and $\mathtt{x} = (Z, C, \hat{C}, T, \mathsf{ivk}, \mathsf{ctx}, m)$, and submit $(\mathtt{Signature}, \mathsf{sid}, \sigma, T, s, \mathsf{tid})$.
         Finally, output $(\mathtt{Ver.Ok}, \mathsf{sid}, \mathsf{ivk}', \mathsf{ctx}, m, \sigma, T, 1)$ to $\mathcal{F}_{\mathsf{SyRA}}$ and record it.
         – Else, the simulation *fails*. We later show that the probability of $\mathcal{A}$ forging a valid signature on behalf of an honest party, or generating a valid signature for a new party that never existed in the system before, is negligible.
         Also, consider the following two bad events: for every two signatures σ_1 and σ_2 with associated T_1 and T_2, (1) If $T_1 = T_2$ and $s_1 \neq s_2$, or (2) If $T_1 \neq T_2$, and $s_1 = s_2$. We will show that the probability of the these bad events is zero.
2. Corrupted verifier V:
   Output to $\mathcal{Z}$ whatever $\mathcal{A}$ outputs.

5.2 Sequence of Games and Reductions

In a series of games, we demonstrate that the two random variables $\mathsf{EXEC}_{\Pi_{\mathsf{SyRA}}, \mathcal{A}, \mathcal{Z}}$ and $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{SyRA}}, \mathcal{S}, \mathcal{Z}}$ are statistically close. We use the notation $\Pr[\mathsf{Game}^{(i)}(k) = 1]$ to represent the likelihood that the environment $\mathcal{Z}$ produces an outcome of 1 in $\mathsf{Game}^{(i)}$. Each specific game $\mathsf{Game}^{(i)}$ is associated with its own ideal functionality $\mathcal{F}^{(i)}_{\mathsf{SyRA}}$ and simulator $\mathcal{S}^{(i)}$. We begin with the least secure (most leaky)

functionality $\mathcal{F}_{\mathsf{SyRA}}^{(0)}$ and corresponding simulator $\mathcal{S}^{(0)}$, and progressively move towards our primary functionality $\mathcal{F}_{\mathsf{SyRA}}$ and primary simulator $\mathcal{S}$.

Let q_t denote the upper bound on the number of signatures of all honest parties and let q_h denote the upper bound on the number of queries the adversary can make to the random oracle, which are both polynomial in the security parameter k.

Summary of Games. Here, we provide an overview of six distinct games $\mathsf{Game}^{(0)}, \ldots, \mathsf{Game}^{(5)}$, where $\mathsf{Game}^{(0)}$ corresponds to the real-world execution $\mathsf{EXEC}_{\Pi_{\mathsf{SyRA}},\mathcal{A},\mathcal{Z}}$ and $\mathsf{Game}^{(5)}$ corresponds to the ideal-world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{SyRA}},\mathcal{S},\mathcal{Z}}$. We provide detailed explanations of the games and security reductions in Section 5.3. Note that the signatures of honest parties are simulated by the simulator $\mathcal{S}$, who emulates $\mathcal{F}_{\mathsf{NIZK}}$.

$\underline{\mathsf{Game}^{(1)}}$: All secret key pairs $(\mathsf{usk}, \hat{\mathsf{usk}})$ of honest parties with corresponding ElGamal ciphertexts $(C, \hat{C})$ are changed to $(A, \hat{B})$, where $A \xleftarrow{\$} \mathbb{G}, \hat{B} \xleftarrow{\$} \hat{\mathbb{G}}$. All values $T = e(Z, \hat{\mathsf{usk}})$ remain the same, and all q_t signatures are with respect to $(A, \hat{B})$. Thus, under the IND-CPA security of ElGamal encryption, we show:

$$|\Pr[\mathsf{Game}^{(1)}(k) = 1] - \Pr[\mathsf{Game}^{(0)}(k) = 1]| \leq 2q_t \cdot \mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}}(k)$$

$\underline{\mathsf{Game}^{(2)}}$: All pseudonyms $T = e(Z, \hat{\mathsf{usk}})$ of honest parties are changed $T = e(Z, \hat{D})$, where $\hat{D} \xleftarrow{\$} \hat{\mathbb{G}}$, and all q_t signatures are computed with these values. Note that in this game, the simulator consistently employs an identical random group element $\hat{D}$ for a given identity string s holder. Under the pseudorandomness of $\mathsf{DY\text{-}VRF}^{\dagger}$, we show:

$$|\Pr[\mathsf{Game}^{(2)}(k) = 1] - \Pr[\mathsf{Game}^{(1)}(k) = 1]| \leq q_t \cdot \mathsf{Adv}^{\text{pseudo}}_{\mathcal{A},\mathsf{DY\text{-}VRF}^{\dagger}}(k)$$

$\underline{\mathsf{Game}^{(3)}}$: If a particular identity string s holder is directed to sign distinct contexts, e.g. $T = (\mathsf{h}(\mathsf{ctx}), \hat{D})$ and $T' = (\mathsf{h}(\mathsf{ctx}'), \hat{D})$, the simulator substitutes distinct random group elements for these values, e.g., $T', T' \xleftarrow{\$} \mathbb{G}_T$. Under the DDH assumption, we show:

$$|\Pr[\mathsf{Game}^{(3)}(k) = 1] - \Pr[\mathsf{Game}^{(2)}(k) = 1]| \leq q_t \cdot q_h \cdot \mathsf{Adv}_{\mathcal{A}}^{\mathsf{DDH}}(k)$$

$\underline{\mathsf{Game}^{(4)}}$: $\mathcal{F}_{\mathsf{SyRA}}^{(4)}$ prohibits $\mathcal{S}^{(4)}$ from submitting any message to $\mathcal{F}_{\mathsf{SyRA}}^{(4)}$ on behalf of the adversary $\mathcal{A}$ (corrupted party) who generates a valid signature for a new party (not belonging to the group of honest and corrupted parties) or generates a valid signature on behalf of an honest party. Under the pseudorandomness of $\mathsf{DY\text{-}VRF}^{\dagger}$, we show:

$$|\Pr[\mathsf{Game}^{(4)}(k) = 1] - \Pr[\mathsf{Game}^{(3)}(k) = 1]| \leq \mathsf{Adv}^{\text{pseudo}}_{\mathcal{A},\mathsf{DY\text{-}VRF}^{\dagger}}(k)$$

$\underline{\mathsf{Game}^{(5)}}$: $\mathcal{F}_{\mathsf{SyRA}}^{(5)}$ does not allow $\mathcal{S}^{(5)}$ to submit any message to $\mathcal{F}_{\mathsf{SyRA}}^{(5)}$ on behalf of the adversary $\mathcal{A}$ (corrupted party) who generates two valid signatures with

one unique identifier s on a given context where $T_1 \neq T_2$, or generates two valid signatures using two identifiers $s_1 \neq s_2$ where $T_1 = T_2$. It is argued that the probability of such events is zero. Hence:

$$\Pr[\mathsf{Game}^{(5)}(k) = 1] = \Pr[\mathsf{Game}^{(4)}(k) = 1]$$

Thus, any PPT environment $\mathcal{Z}$ can distinguish $\mathsf{EXEC}_{\Pi_{\mathsf{SyRA}},\mathcal{A},\mathcal{Z}}$ from $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{SyRA}},\mathcal{S},\mathcal{Z}}$ with probability bounded by:

$$2q_t \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}}(k) + (q_t + 1) \cdot \mathsf{Adv}^{\mathrm{pseudo}}_{\mathcal{A},\mathsf{DY\text{-}VRF}^\dagger}(k) + q_t \cdot q_h \cdot \mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{A}}(k).$$

5.3 Details of Games and Associated Reductions

We now provide a detailed description of the games and reductions in our security proof.

Remark 1. For the sake of simplicity and to prevent redundancy, we will refrain from including details like channel leakages in the following games. They have already been discussed in the simulator description in Section 5.1.

$\underline{\mathsf{Game}^{(0)}}$: Initially, $\mathcal{F}^{(0)}_{\mathsf{SyRA}}$ forwards all communication with $\mathcal{Z}$. The simulator $\mathcal{S}^{(0)}$ corresponds to the execution of the real-world protocol $\mathsf{EXEC}_{\Pi_{\mathsf{SyRA}},\mathcal{A},\mathcal{Z}}$.

$\underline{\mathsf{Game}^{(1)}}$: Same as $\mathsf{Game}^{(0)}$, except that in $\mathsf{Game}^{(1)}$, we change all honest parties' plaintexts $(\mathsf{usk}, \hat{\mathsf{usk}})$ of encryptions $(C, \hat{C})$ to random group elements selected from $\mathbb{G}$ and $\hat{\mathbb{G}}$, respectively. However, note that T values are computed with real-world values.

We show that $\mathsf{Game}^{(1)}$ and $\mathsf{Game}^{(0)}$ are indistinguishable under the $\mathsf{IND\text{-}CPA}$ security of ElGamal encryption, which allows us to bound the probability that $\mathcal{Z}$ distinguishes $\mathsf{Game}^{(1)}$ from $\mathsf{Game}^{(0)}$.

We introduce a sequences of sub-games:

$$(\mathsf{Game}^{(0)}_0 := \mathsf{Game}^{(0)}, \ldots, \mathsf{Game}^{(0)}_{i-1}, \mathsf{Game}^{(0)}_i, \ldots, \mathsf{Game}^{(0)}_{2q_t} := \mathsf{Game}^{(1)})$$

where q_t denotes the upper bound on the number of all signatures of all honest parties. Define $\mathsf{Game}^{(0)}_0 := \mathsf{Game}^{(0)}$ as the real-world execution $\mathsf{EXEC}_{\Pi_{\mathsf{SyRA}},\mathcal{A},\mathcal{Z}}$. $\mathsf{Game}^{(0)}_1$ is the same as $\mathsf{Game}^{(0)}_0$, except we change the plaintext of the first ciphertext of the first honest party from the real-world value to the ideal-world random group element. Finally, we do the same for the last (second) ciphertext of the last honest party such that in $\mathsf{Game}^{(0)}_{2q_t} := \mathsf{Game}^{(1)}$, all ciphertexts are generated from random group elements by $\mathcal{S}^{(0)}_{2q_t} = \mathcal{S}^{(1)}$. The reduction between $\mathsf{Game}^{(0)}_{i-1}$ and $\mathsf{Game}^{(0)}_i$ is described below, where any difference between them is upper bounded by $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}}(k)$.

In leaky functionalities $\mathcal{F}^{(0)}_{\mathsf{SyRA},i}, 1 \leq i \leq 2q_t$, where $\mathcal{F}^{(0)}_{\mathsf{SyRA},1} := \mathcal{F}^{(0)}_{\mathsf{SyRA}}$ and $\mathcal{F}^{(0)}_{\mathsf{SyRA},2q_t} := \mathcal{F}^{(1)}_{\mathsf{SyRA}}$, the leaked message to the simulator in the `Issue` and `Sign` commands are $(\mathtt{Issue}, \mathsf{sid}, \mathsf{id}, \mathsf{P}, s, \mathtt{x}_s)$ and $(\mathtt{Sign}, \mathsf{sid}, \mathsf{P}, \mathsf{ctx}, m, \mathsf{tid})$, respectively,

for honest parties. Hence, the simulator knows (P, s) values for all honest parties. Later, in subsequent games, the associated functionalities become less leaky and closer to our main functionality $\mathcal{F}_{\mathsf{SyRA}}$.

Associated Reduction Between $\mathsf{Game}_{i-1}^{(0)}$ and $\mathsf{Game}_{i}^{(0)}$ ($\mathsf{IND\text{-}CPA}$ security of ElGamal encryption; for $1 \leq i \leq 2q_t$). If $\mathcal{Z}$ distinguishes $\mathsf{Game}_{i-1}^{(0)}$ and $\mathsf{Game}_{i}^{(0)}$, we can construct $\mathcal{A}'$ that breaks the $\mathsf{IND\text{-}CPA}$ security of ElGamal encryption used in our construction.

- For $1 \leq j \leq i-1$: all (real-world) plaintexts have already been substituted with (ideal-world) random values.
- For $i+1 \leq j \leq 2q_t$: all ciphertexts are created using real-world plaintexts.
- The challenger $\mathcal{C}$ of the $\mathsf{IND\text{-}CPA}$ game outputs pk to $\mathcal{A}'$.
- $\mathcal{A}'$ sets $W \leftarrow \mathsf{pk}$ in ivk. For simplicity, we omit writing the details for $\hat{C}$, as it is similar to C. (In our construction, pk for ElGamal ciphertext C is $W \in \mathbb{G}$ and for $\hat{C}$ it is $\hat{W} \in \hat{\mathbb{G}}$.)
- The real-world plaintext value $g^{1/(s+\mathsf{isk})}$ (associated to $b=0$ in the $\mathsf{IND\text{-}CPA}$ game) is changed to A in $\mathsf{Game}_{i}^{(0)}$, where $A \xleftarrow{\$} \mathbb{G}$ (which is the ideal-world value associated to $b=1$ in the $\mathsf{IND\text{-}CPA}$ game).
- The challenger provides C_b to distinguisher $\mathcal{A}'$: for $b=0$, C_0 encrypts $g^{1/(s+\mathsf{isk})}$, and for $b=1$, C_1 encrypts A. Therefore, for $\mathcal{Z}$, $\mathsf{Game}_{i}^{(0)}$ is the same as running the real-world protocol with real-world value usk for an honest party, rather than a random group element. Hence, if $\mathcal{Z}$ distinguishes $\mathsf{Game}_{i-1}^{(0)}$ from $\mathsf{Game}_{i}^{(0)}$, $\mathcal{A}'$ uses this to win the $\mathsf{IND\text{-}CPA}$ game.

The next game hop, namely between $\mathsf{Game}_{i}^{(0)}$ and $\mathsf{Game}_{i+1}^{(0)}$, for the value $\hat{\mathsf{usk}}$ of honest parties encrypted to $\hat{C}$ is similar, so we omit the details.

$\mathcal{A}'$ simulates signatures σ of honest parties as follows:

- Leak $(\mathtt{Prove}, \mathsf{sid}, \mathtt{x})$ to $\mathcal{A}$ where $\mathtt{x} = (Z, C, \hat{C}, T, \mathsf{ivk}, \mathsf{ctx}, m)$.
- Upon receiving $(\mathtt{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{A}$, emulate $\mathcal{F}_{\mathsf{NIZK}}$ record $(\pi, \mathtt{x})$.
- Set $\sigma \leftarrow (\pi, \mathtt{x})$ and $\mathcal{L}_{\mathsf{SS}} \leftarrow \mathcal{L}_{\mathsf{SS}} \cup \{\sigma\}$. Recall that $\mathcal{L}_{\mathsf{SS}}$ is the list of simulated signatures.
- Submit $(\mathtt{Signature}, \mathsf{sid}, \sigma, s^*, \mathsf{tid})$ to $\mathcal{F}_{\mathsf{SyRA},i}^{(0)}$ where $s^* = 0$.

Moreover, $\mathcal{A}'$ is able to simulate valid secret keys $(\mathsf{usk}, \hat{\mathsf{usk}})$ of corrupted parties. This is similar to the simulation of the case "*Corrupted party* P *and honest issuer* Iss" in the *Issue* protocol in Section 5.1. Therefore, under the $\mathsf{IND\text{-}CPA}$ security of ElGamal encryption, the following inequality holds:

$$|\Pr[\mathsf{Game}^{(1)}(k)=1] - \Pr[\mathsf{Game}^{(0)}(k)=1]| \leq 2q_t \cdot \mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}}(k)$$

$\underline{\mathsf{Game}^{(2)}}$: Same as $\mathsf{Game}^{(1)}$, except that in $\mathsf{Game}^{(2)}$, we change all tags $T = e(Z, \hat{\mathsf{usk}})$ of honest parties to tags of the form $T = e(Z, \hat{D})$ where $\hat{D} \xleftarrow{\$} \hat{\mathbb{G}}$. We highlight that in this game, for a fixed identifier s, the simulator always uses the *same* random group element $\hat{D}$. Hence, $\mathsf{Game}^{(2)}$ is the same as $\mathsf{Game}^{(1)}$, except

$\mathcal{S}^{(2)}$ computes the T values using random group elements, rather than real-world $\hat{\mathsf{usk}}$ values, for all honest parties.

We show that $\mathsf{Game}^{(2)}$ and $\mathsf{Game}^{(1)}$ are indistinguishable under the pseudorandomness of DY-VRF†, which allows us to bound the probability that $\mathcal{Z}$ distinguishes $\mathsf{Game}^{(2)}$ from $\mathsf{Game}^{(1)}$.

We introduce a sequence of sub-games:

$$(\mathsf{Game}_0^{(1)} := \mathsf{Game}^{(1)}, \ldots, \mathsf{Game}_{j-1}^{(1)}, \mathsf{Game}_j^{(1)}, \ldots, \mathsf{Game}_{q_t}^{(1)} := \mathsf{Game}^{(2)})$$

(where q_t denotes the upper bound on the number of all signatures of all honest parties.) Define $\mathsf{Game}_0^{(1)} := \mathsf{Game}^{(1)}$. $\mathsf{Game}_1^{(1)}$ is the same as $\mathsf{Game}_0^{(1)}$, except in $\mathsf{Game}_1^{(1)}$, we change the tag T from the real-world value $T = e(Z, \hat{\mathsf{usk}})$ to the ideal-world value $T = e(Z, \hat{D})$, where $\hat{D} \xleftarrow{\$} \hat{\mathbb{G}}$. Finally, we do the same for the last tag such that in $\mathsf{Game}_{q_t}^{(1)} := \mathsf{Game}^{(2)}$, all tags are generated using random group elements (instead of real-world values $\hat{\mathsf{usk}}$). The reduction between $\mathsf{Game}_{j-1}^{(1)}$ and $\mathsf{Game}_j^{(1)}$ is described below, where any difference between them is upper bounded by $\mathsf{Adv}^{\text{pseudo}}_{\mathcal{A},\text{DY-VRF}^{\dagger}}(k)$.

In leaky functionalities $\mathcal{F}^{(1)}_{\mathsf{SyRA},j}, 1 \leq j \leq q_t$, where $\mathcal{F}^{(1)}_{\mathsf{SyRA},1} := \mathcal{F}^{(1)}_{\mathsf{SyRA}}$ and $\mathcal{F}^{(1)}_{\mathsf{SyRA},q_t} := \mathcal{F}^{(2)}_{\mathsf{SyRA}}$), the leaked message to the simulator in the `Issue` and `Sign` commands are $(\mathtt{Issue}, \mathsf{sid}, \mathsf{id}, \mathsf{P}, s, \mathtt{x}_s)$ and $(\mathtt{Sign}, \mathsf{sid}, \mathsf{P}, \mathsf{ctx}, m, \mathsf{tid})$, respectively, for an honest party P.

Associated Reduction Between $\mathsf{Game}_{j-1}^{(1)}$ and $\mathsf{Game}_j^{(1)}$ (pseudorandomness of DY-VRF†; for $1 \leq j \leq q_t$). If $\mathcal{Z}$ distinguishes $\mathsf{Game}_{j-1}^{(1)}$ and $\mathsf{Game}_j^{(1)}$, we can construct $\mathcal{A}'$ that breaks the pseudorandomness of DY-VRF†.

- For $1 \leq k \leq j-1$: all (real-world) $\hat{\mathsf{usk}}$ values in tags $T = e(Z, \hat{\mathsf{usk}})$ have already been substituted with (ideal-world) random values $\hat{D} \xleftarrow{\$} \hat{\mathbb{G}}$, $T = e(Z, \hat{D})$.
- For $j+1 \leq k \leq q_t$: all tags are created using real-world $\hat{\mathsf{usk}}$ values.
- The challenger $\mathcal{C}$ of the pseudorandomness game outputs $\hat{\mathsf{IPK}}$ to $\mathcal{A}'$.
- $\mathcal{A}'$ sets $\hat{\mathsf{ivk}} \leftarrow \hat{\mathsf{IPK}}$.
- Program random oracle as: $Z_l = \mathsf{h}(m_{1l}) \leftarrow g^{r_l}$ for $r_l \xleftarrow{\$} \mathbb{Z}_p^*$.
- Upon receiving $(\mathtt{Sign}, \mathsf{sid}, \mathsf{P}, \mathsf{ctx}, m, \mathsf{tid})$ from $\mathcal{F}^{(1)}_{\mathsf{SyRA},j}$, $\mathcal{A}'$ retrieves the associated s value of P using the leaked message $(\mathtt{Issue}, \mathsf{sid}, \mathsf{id}, \mathsf{P}, s, \mathtt{x}_s)$.
- If there exists an entry (s', r'_s) recorded where $s' = s$, set $\hat{D} \leftarrow \hat{g}^{r'_s}$. Else, pick $r_s \xleftarrow{\$} \mathbb{Z}_p^*$. Record (s, r_s), and set $\hat{D} \leftarrow \hat{g}^{r_s}$.
- The real-world tag value $T_{\mathsf{real}} = e(g, \hat{g}^{1/(s+\mathsf{isk})})^{r_l}$ (associated to $b = 0$ in the pseudorandomness game) is changed to the ideal-world value $T_{\mathsf{ideal}} = e(g, \hat{D})^{r_l}$ in $\mathsf{Game}_i^{(0)}$ (associated to $b = 1$ in the pseudorandomness game).
- $\mathcal{A}'$ provides s (for which $s \notin \mathbb{S}^{\mathsf{corrupted}}$ holds) to the challenger $\mathcal{C}$ (where $s \notin \mathcal{Q}$; $\mathcal{Q}$ is a query list maintained by $\mathcal{C}$, which as we will see only contains s values of corrupted parties.).

– Upon receiving the challenge target group element T_b given by $\mathcal{C}$ to distinguisher $\mathcal{A}'$, for case $b = 0$: $T_{\mathsf{real}} = T_0$; and for $b = 1$: $T_{\mathsf{ideal}} = T_1$. If $\mathcal{Z}$ distinguishes $\mathsf{Game}^{(1)}_{j-1}$ from $\mathsf{Game}^{(1)}_{j}$, $\mathcal{A}'$ uses this to win the pseudorandomness game. Therefore, for $\mathcal{Z}$, $\mathsf{Game}^{(1)}_{j}$ is the same as running the real-world protocol with real-world $\hat{\mathsf{usk}}$ value used in tag computation for an honest party, rather than a random group element from $\hat{\mathbb{G}}$.

$\mathcal{A}'$ simulates signatures σ of honest parties as follows:

– Leak $(\mathtt{Prove}, \mathsf{sid}, \mathtt{x})$ to $\mathcal{A}$ where $\mathtt{x} = (Z, C, \hat{C}, T, \mathsf{ivk}, \mathsf{ctx}, m)$.
– Upon receiving $(\mathtt{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{A}$, emulate $\mathcal{F}_{\mathsf{NIZK}}$ record $(\pi, \mathtt{x})$.
– Set $\sigma \leftarrow (\pi, \mathtt{x})$ and $\mathcal{L}_{\mathsf{SS}} \leftarrow \mathcal{L}_{\mathsf{SS}} \cup \{\sigma\}$.
– Submit $(\mathtt{Signature}, \mathsf{sid}, \sigma, s^*, \mathsf{tid})$ to $\mathcal{F}^{(1)}_{\mathsf{SyRA},j}$ where $s^* = 0$.

Moreover, $\mathcal{A}'$ is able to simulate valid secret keys $(\mathsf{usk}, \hat{\mathsf{usk}})$ of corrupted parties (different from the previous game, without knowing isk) as follows:

– Upon receiving the message $(\mathtt{Send}, \mathsf{sid}, \mathsf{lss}, (s, \mathtt{x}_s, \pi_s))$ from $\mathcal{A}$ submitted to $\mathcal{F}^{\mathsf{SSA}}_{\mathsf{Ch}}$, $\mathcal{A}'$ acts as follows:
– Provide $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}_s, \pi_s)$ to $\mathcal{A}$.
– Upon receiving $(\mathtt{Witness}, \mathsf{sid}, (s, \mathtt{w}_s))$ from $\mathcal{A}$, submit $(\mathtt{Issue}, \mathsf{sid}, s, \mathtt{x}_s, \mathtt{w}_s)$ on behalf of (corrupted) P to $\mathcal{F}^{(1)}_{\mathsf{SyRA},j}$.
– Upon receiving $(\mathtt{Issue}, \mathsf{sid}, \mathsf{id}, \mathtt{x}_s)$ from $\mathcal{F}^{(1)}_{\mathsf{SyRA},j}$, $\mathcal{A}'$ queries $\mathcal{C}$ using s provided by $\mathcal{A}$.
– $\mathcal{C}$ sets $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{s\}$ and sends $g^{1/(s+\mathsf{isk})}$ and $\hat{g}^{1/(s+\mathsf{isk})}$ to $\mathcal{A}'$.
– Upon receiving $g^{1/(s+\mathsf{isk})}$ and $\hat{g}^{1/(s+\mathsf{isk})}$ from $\mathcal{C}$, $\mathcal{A}'$ sets $\mathsf{usk} \leftarrow g^{1/(s+\mathsf{isk})}$ and $\hat{\mathsf{usk}} \leftarrow \hat{g}^{1/(s+\mathsf{isk})}$.
– Set $\mathcal{L}_{\mathsf{CR}} \leftarrow \mathcal{L}_{\mathsf{CR}} \cup \{\mathsf{P}\}$, and $\mathbb{S}^{\mathsf{corrupted}} \leftarrow \mathbb{S}^{\mathsf{corrupted}} \cup \{s\}$. Recall that $\mathcal{L}_{\mathsf{CR}}$ is the list of corrupted-registered parties (who have been issued signing keys).
– Provide $(\mathtt{Received}, \mathsf{sid}, \mathsf{lss}, (\mathsf{usk}, \hat{\mathsf{usk}}))$ to $\mathcal{A}$.
– Submit $(\mathtt{Issued}, \mathsf{sid}, \mathsf{id})$ to $\mathcal{F}^{(1)}_{\mathsf{SyRA},j}$, and output to $\mathcal{Z}$ whatever $\mathcal{A}$ outputs.

Therefore, under the pseudorandomness of $\mathsf{DY\text{-}VRF}^{\dagger}$, the following inequality holds:

$$|\Pr[\mathsf{Game}^{(2)}(k) = 1] - \Pr[\mathsf{Game}^{(1)}(k) = 1]| \leq q_t \cdot \mathsf{Adv}^{\mathrm{pseudo}}_{\mathcal{A},\mathsf{DY\text{-}VRF}^{\dagger}}(k)$$

$\underline{\mathsf{Game}^{(3)}}$: Same as $\mathsf{Game}^{(2)}$, except that in $\mathsf{Game}^{(3)}$, we change all tags $T = e(Z, \hat{D})$ of honest parties to random values $T \xleftarrow{\$} \mathbb{G}_T$. We highlight that in this game, for a fixed identifier s, the simulator samples fresh random group elements for different contexts unless the same identity string s holder is instructed to sign the same context. (In this case, the simulator receives the same ucid from the ideal functionality and uses the same tag value T, as we will see.)

Specifically, in $\mathsf{Game}^{(2)}$, if the same s holder was instructed to sign different contexts, the simulator used the same $\hat{D}$ value for tags (e.g., $T_1 = e(\mathsf{h}(\mathsf{ctx}_1), \hat{D})$

and $T_2 = e(\mathsf{h}(\mathsf{ctx}_2), \hat{D})$, where $\hat{D} \xleftarrow{\$} \hat{\mathbb{G}}$, and h is the output of $\mathcal{F}_{\mathsf{RO}}$). In $\mathsf{Game}^{(3)}$, if the same s holder is instructed to sign different contexts, the simulator uses different random group elements for tag computation: $(T_1, T_2) \xleftarrow{\$} \mathbb{G}_T^2$.

In leaky functionalities $\mathcal{F}^{(2)}_{\mathsf{SyRA},j}, 1 \leq j \leq q_t$, where $\mathcal{F}^{(2)}_{\mathsf{SyRA},1} := \mathcal{F}^{(2)}_{\mathsf{SyRA}}$ and $\mathcal{F}^{(2)}_{\mathsf{SyRA},q_t} := \mathcal{F}^{(3)}_{\mathsf{SyRA}}$), the leaked message to the simulator in the `Issue` and `Sign` commands $(\mathtt{Issue}, \mathsf{sid}, \mathsf{id}, \mathsf{P}, \mathtt{x}_s)$ and $(\mathtt{Sign}, \mathsf{sid}, \mathsf{P}, \mathsf{ctx}, m, \mathsf{tid})$, respectively, for an honest party P.

We show that $\mathsf{Game}^{(3)}$ and $\mathsf{Game}^{(2)}$are indistinguishable under the DDH assumption, which allows us to bound the probability that $\mathcal{Z}$ distinguishes $\mathsf{Game}^{(3)}$ from $\mathsf{Game}^{(2)}$.

We introduce a sequences of sub-games:

$$(\mathsf{Game}^{(2)}_0 := \mathsf{Game}^{(2)}, \ldots, \mathsf{Game}^{(2)}_{j-1}, \mathsf{Game}^{(2)}_j, \ldots, \mathsf{Game}^{(2)}_{q_t} := \mathsf{Game}^{(3)})$$

(where q_t denotes the upper bound on the number of all signatures of all honest parties.) Define $\mathsf{Game}^{(2)}_0 := \mathsf{Game}^{(2)}$. $\mathsf{Game}^{(2)}_1$ is the same as $\mathsf{Game}^{(2)}_0$, except in $\mathsf{Game}^{(2)}_1$, we change the *first* tag value of the *first* honest party (e.g., who holds s_1) from $T_{1,1} = e(Z, \hat{D})$ to $T_{1,1} \xleftarrow{\$} \mathbb{G}_T$. in $\mathsf{Game}^{(2)}_2$, we change the *second* tag value of the *first* honest party (holding s_1) from $T_{2,1} = e(Z', \hat{D})$ to $T_{2,1} \xleftarrow{\$} \mathbb{G}_T$, and so on. The reduction between $\mathsf{Game}^{(2)}_{j-1}$ and $\mathsf{Game}^{(2)}_j$ is described below, where any difference between them is upper bounded by $\mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{A}}(k)$.

Associated Reduction Between $\mathsf{Game}^{(2)}_{j-1}$ and $\mathsf{Game}^{(2)}_j$ (DDH; for $1 \leq j \leq q_t$). If $\mathcal{Z}$ distinguishes $\mathsf{Game}^{(2)}_{j-1}$ and $\mathsf{Game}^{(2)}_j$, we can construct $\mathcal{A}'$ that breaks the DDH assumption.

- For $1 \leq k \leq j-1$: all $\hat{D}$ values (associated to each honest party P) in tags $T = e(Z, \hat{D})$ have already been substituted. For instance, $T_1 = e(Z, \hat{D})$ and $T_2 = e(Z', \hat{D})$ associated with an honest party have changed to $(T_1, T_2) \xleftarrow{\$} \mathbb{G}_T^2$.
- For $j+1 \leq k \leq q_t$: all tags are created such that for a fixed s holder the simulator always uses the same group element $\hat{D}$ in tag computation.
- The challenger $\mathcal{C}$ of DDH for $g \in \mathbb{G}$ computes: $X = g^x, Y = g^y, C_0 = g^{xy}$, and $C_1 = g^r$ where $\mathcal{C}$ samples (x, y, r) randomly: $(x, y, r) \xleftarrow{\$} \mathbb{Z}_p^*$.
- The challenger $\mathcal{C}$ provides (g, X, Y, C_b) to $\mathcal{A}'$ where $b = 0$ or $b = 1$.
  Let q_h be the total number of queries the adversary can make to the random oracle. Then:
- $i \xleftarrow{\$} [1, q_h]$, $\mathcal{A}'$ sets $Z = \mathsf{h}(\mathsf{ctx}) \leftarrow X$ if $\mathtt{ctr} = i$; ($\mathtt{ctr}$ is initially set to zero). Else, program the random oracle as: $Z_k = \mathsf{h}(\mathsf{ctx}_k) \leftarrow g^{x_k}$ where $x_k \xleftarrow{\$} \mathbb{Z}_p^*$ and set $\mathtt{ctr} \leftarrow \mathtt{ctr} + 1$.
  The tag value $T = e(\mathsf{h}(\mathsf{ctx}), \hat{D})$, which is computed as $e(C_0, \hat{g})$ (associated to $b = 0$ in the DDH game), is changed to $T = e(C_1, \hat{g})$ (associated to $b = 1$ in the DDH game) in $\mathsf{Game}^{(2)}_j$. Therefore, if $\mathcal{Z}$ distinguishes $\mathsf{Game}^{(2)}_{j-1}$ from $\mathsf{Game}^{(2)}_j$, $\mathcal{A}'$ uses this to win the DDH game.

$\mathcal{A}'$ simulates signatures σ of honest parties as follows:

- Leak $(\texttt{Prove}, \mathsf{sid}, \mathtt{x})$ to $\mathcal{A}$ where $\mathtt{x} = (Z, C, \hat{C}, T, \mathsf{ivk}, \mathsf{ctx}, m)$.
- Upon receiving $(\texttt{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{A}$, emulate $\mathcal{F}_{\mathsf{NIZK}}$ record $(\pi, \mathtt{x})$.
- Set $\sigma \leftarrow (\pi, \mathtt{x})$ and $\mathcal{L}_{\mathsf{SS}} \leftarrow \mathcal{L}_{\mathsf{SS}} \cup \{\sigma\}$.
- Submit $(\texttt{Signature}, \mathsf{sid}, \sigma, s^*, \mathsf{tid})$ to $\mathcal{F}^{(2)}_{\mathsf{SyRA},j}$ where $s^* = 0$.

Moreover, $\mathcal{A}'$ is able to simulate valid secret keys $(\mathsf{usk}, \hat{\mathsf{usk}})$ of corrupted parties. This is similar to the simulation of the case "*Corrupted party* P *and honest issuer* Iss" in the *Issue* protocol in Section 5.1. Therefore, under the DDH assumption, the following inequality holds:

$$|\Pr[\mathsf{Game}^{(3)}(k) = 1] - \Pr[\mathsf{Game}^{(2)}(k) = 1]| \leq q_t \cdot q_h \cdot \mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{A}}(k)$$

$\underline{\mathsf{Game}^{(4)}}$: Same as $\mathsf{Game}^{(3)}$, except that in $\mathsf{Game}^{(4)}$, $\mathcal{F}^{(4)}_{\mathsf{SyRA}}$ does not allow $\mathcal{S}^{(4)}$ to submit any message to $\mathcal{F}^{(4)}_{\mathsf{SyRA}}$ on behalf of adversary $\mathcal{A}$ (corrupted party) who generates a valid signature for a new party (who is not among honest and corrupted parties) or who generates a valid signature on behalf of an honest party. Hence, $\mathsf{Game}^{(4)}$ is the same as $\mathsf{Game}^{(3)}$, except that it checks whether a flag is raised or not. The flag is raised if $\mathcal{A}$ can generate a valid signature for a completely new party, who is neither honest nor corrupted, or forge a valid signature for an honest party.

We show that $\mathsf{Game}^{(4)}$ and $\mathsf{Game}^{(3)}$ are indistinguishable under the pseudorandomness of $\mathsf{DY\text{-}VRF}^{\dagger}$, which allows us to bound the probability that $\mathcal{Z}$ distinguishes $\mathsf{Game}^{(4)}$ from $\mathsf{Game}^{(3)}$.

Note that $\mathcal{F}^{(4)}_{\mathsf{SyRA}}$, in the `Issue` and `Sign` commands, does not leak the identity of the (honest) party P to $\mathcal{S}^{(4)}$. $\mathcal{F}^{(4)}_{\mathsf{SyRA}}$ leaks to ideal-world adversary $\mathcal{S}$ whatever our final $\mathcal{F}_{\mathsf{SyRA}}$ leaks to $\mathcal{S}$, which is $(\texttt{Sign}, \mathsf{sid}, \mathsf{ucid}, \mathsf{ctx}, m, \mathsf{tid})$. As a result, $\mathcal{S}^{(4)}$ picks a random group element as the tag value, except in the instance that the same ucid is received from the functionality.

Associated Reduction Between $\mathsf{Game}^{(4)}$ and $\mathsf{Game}^{(3)}$ (pseudorandomness of $\mathsf{DY\text{-}VRF}^{\dagger}$). If $\mathcal{Z}$ distinguishes $\mathsf{Game}^{(4)}$ and $\mathsf{Game}^{(3)}$, we can construct $\mathcal{A}'$ that breaks the pseudorandomness of $\mathsf{DY\text{-}VRF}^{\dagger}$.

The final simulator $\mathcal{S}$ is described in Section 5.1. To prevent redundancy, we will highlight distinctions from this simulator. $\mathcal{S}^{(4)}$ fails with probability at most $\mathsf{Adv}^{\mathrm{pseudo}}_{\mathcal{A},\mathsf{DY\text{-}VRF}^{\dagger}}(k)$ as it is shown below in cases where emulating honest verifier $\mathcal{S}^{(4)}$ receives a valid signature (from $\mathcal{A}$) that is never simulated by $\mathcal{S}^{(4)}$.

- Given the extracted witness $\mathtt{w} = (s, \alpha, \beta)$ from $\mathcal{A}$'s statement and proof $(\mathtt{x}, \pi)$ (see Section 5.1 for more details), $\mathcal{A}'$ provides $s \notin \mathbb{S}^{\mathsf{corrupted}}$ to the challenger $\mathcal{C}$ of the pseudorandomness game (where $s \notin \mathcal{Q}$; recall that $\mathcal{Q}$ is a query list maintained by $\mathcal{C}$, which only contains s values of corrupted parties).
- The challenger $\mathcal{C}$ provides T_b to $\mathcal{A}'$.
- Parse $\mathtt{x} = (Z, C, \hat{C}, T, \mathsf{ivk}, \mathsf{ctx}, m)$. For case $b = 0$: $T_0 = T$; and for $b = 1$: $T_1 \neq T$. Hence, if $\mathcal{A}$ provides a valid signature (see Section 5.1 for a comprehensive

description of the steps involved in verifying the signature) for $s \notin \mathbb{S}^{\mathsf{corrupted}}$, $\mathcal{A}'$ uses that to win the pseudorandomness game. Therefore, the probability of failure for $\mathcal{S}^{(4)}$ is upper bounded by $\mathsf{Adv}^{\text{pseudo}}_{\mathcal{A},\mathsf{DY\text{-}VRF}^\dagger}(k)$.

Simulating the view of $\mathcal{A}$ for the valid keys $(\mathsf{usk}, \hat{\mathsf{usk}})$ of corrupted parties (without knowing isk) is as in $\mathsf{Game}^{(2)}$. $\mathcal{A}'$ simulates the signatures of honest users as in the previous game. Therefore, under the pseudorandomness of $\mathsf{DY\text{-}VRF}^\dagger$, the following inequality holds:

$$|\Pr[\mathsf{Game}^{(4)}(k) = 1] - \Pr[\mathsf{Game}^{(3)}(k) = 1]| \leq \mathsf{Adv}^{\text{pseudo}}_{\mathcal{A},\mathsf{DY\text{-}VRF}^\dagger}(k)$$

$\underline{\mathsf{Game}^{(5)}}$: Same as $\mathsf{Game}^{(4)}$, except that in $\mathsf{Game}^{(5)}$, $\mathcal{F}^{(5)}_{\mathsf{SyRA}}$ does not allow $\mathcal{S}^{(5)}$ to submit any message to $\mathcal{F}^{(5)}_{\mathsf{SyRA}}$ on behalf of adversary $\mathcal{A}$ (corrupted party) who:

- generates two valid signatures on behalf of a corrupted party on a context where $T_1 \neq T_2$; or
- generates two valid signatures on a context using two real-world identifiers $s_1 \neq s_2$ where $T_1 = T_2$.

Hence, $\mathsf{Game}^{(5)}$ is the same as $\mathsf{Game}^{(4)}$, except that it checks whether a flag is raised or not. The flag is raised if one of the two events above occurs. Hence, the probability that $\mathcal{Z}$ distinguishes $\mathsf{Game}^{(5)}$ from $\mathsf{Game}^{(4)}$ is zero, as tag values have uniqueness with respect to the context and s value of the party. More precisely:

- For a given s and context ctx, all generated signatures have the same tag $T = e(Z, \hat{g})^{1/(\mathsf{isk}+s)}$.
- For a given context ctx, s_1, and s_2 where $s_1 \neq s_2$, we always have $T_1 = e(Z, \hat{g})^{1/(\mathsf{isk}+s_1)} \neq T_2 = e(Z, \hat{g})^{1/(\mathsf{isk}+s_2)}$, as $s \ll |\mathbb{Z}^*_p|$.

Hence, the flag is never raised, and the following equality holds:

$$|\Pr[\mathsf{Game}^{(5)}(k) = 1] = \Pr[\mathsf{Game}^{(4)}(k) = 1]$$

As $\mathcal{F}^5_{\mathsf{SyRA}} = \mathcal{F}_{\mathsf{SyRA}}$ and $\mathcal{S}^5 = \mathcal{S}$, so that $\mathsf{Game}^{(5)}$ corresponds to the ideal-world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{SyRA}},\mathcal{S},\mathcal{Z}}$, we argue that random variables $\mathsf{EXEC}_{\Pi_{\mathsf{SyRA}},\mathcal{A},\mathcal{Z}}$ and $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{SyRA}},\mathcal{S},\mathcal{Z}}$ are statistically close. Indeed, the probability for any PPT environment $\mathcal{Z}$ to distinguish $\mathsf{EXEC}_{\Pi_{\mathsf{SyRA}},\mathcal{A},\mathcal{Z}}$ from $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{SyRA}},\mathcal{S},\mathcal{Z}}$ is upper bounded by

$$2q_t \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}}(k) + (q_t + 1) \cdot \mathsf{Adv}^{\text{pseudo}}_{\mathcal{A},\mathsf{DY\text{-}VRF}^\dagger}(k) + q_t \cdot q_h \cdot \mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{A}}(k)$$

which concludes the security proof.

6 Threshold Sybil-Resilient Anonymous Signatures

Recall that our construction of a Sybil-resilient anonymous signature scheme (Section 4) consists of two main parts: (1) issuance of each user's secret key as

a function of their unique identity string s and the issuer secret key, and (2) generation of Sybil-resilient anonymous signatures by users under the obtained secret keys. In this section, we show how the first part may be extended to allow distributed computation of user secret keys. This is desirable for a number of reasons, including the following:

1. *Distributing trust.* A single trusted issuer represents a single point of attack and failure.
2. *Robustness.* Allowing a threshold number of issuers to collectively sign a user's identity string s allows some number of them to be offline, as long as the threshold is reached.
3. *Confidentiality.* A user's identity string s may be private or sensitive information, such as a Social Security Number (SSN) in the US. No single issuer ever sees the user's string in the clear, and a threshold number of them would need to collude to break privacy.

Threshold Sybil-resilient anonymous signatures allow multiple issuers, each possessing a share of the signing key isk, to issue a key pair $(\mathsf{usk}, \hat{\mathsf{usk}})$ to each user in the system. In particular, we consider a set of n issuers, each in possession of shares $(\mathsf{ivk}_i, \mathsf{isk}_i)$. Recall that in our single-issuer construction, a user secret key $(\mathsf{usk}, \hat{\mathsf{usk}})$ is computed as $(g^{1/(s+\mathsf{isk})}, \hat{g}^{1/(s+\mathsf{isk})})$, which has the form of two Dodis-Yampolskiy VRF proofs [DY05], one in each source group (Fig. 4). As noted in the original paper, it is straightforward to construct a distributed computation of the function $f_{\mathsf{isk}}(s) = g^{1/(s+\mathsf{isk})}$ when the issuers have shares of the secret isk. Indeed, there are well-known techniques for multi-party addition, inversion, and exponentiation [Bea92,KPR18], which we specify below for completeness.

In a threshold Sybil-resilient anonymous signature scheme, the public parameters are generated via a distributed secret sharing $\mathsf{PP} \leftarrow \mathsf{T.Setup}$ and given as input to all other algorithms and protocols. For generation of the issuer keys, a distributed key generation protocol (DKG) is used, which outputs the joint public key ivk representing the set of n issuers as well as n verification key shares $\{\mathsf{ivk}_i\}_{i\in[n]}$ and secret key shares $\{\mathsf{isk}_i\}_{i\in[n]}$, one held by each issuer. To collectively "sign" a user's identity string s, at least a threshold of t issuers engage in an interactive protocol $\mathsf{T.Issue}$. At the end of the protocol, the issuers' individual shares of the user secret key are combined by the user to form the key pair $(\mathsf{usk}, \hat{\mathsf{usk}})$, which the user can verify as in the single-party construction. Signature generation and verification are the same as in Section 4.

Efficient and simulatable distributed key generation. In our threshold construction, we make repeated use of the Gennaro et al. [GJKR99] distributed key generation protocol $\mathsf{New.DKG}(t, n)$ (Fig. 5). It can be viewed as n parallel instantiations of Pedersen verifiable secret sharing ($\mathsf{Pedersen\text{-}VSS}$) [Ped92], which is derived from Shamir secret sharing [Sha79] but additionally requires each participant to provide a vector of Pedersen commitments to ensure their received share is consistent with all other participants' shares. It is well known that some popular alternatives, such as the Pedersen DKG [Ped92], do not produce a uniformly random joint public key [GJKR99], whereas the Gennaro et al. DKG is

fully simulatable and therefore compatible with threshold constructions in the UC setting.

Construction.

1. $\mathsf{PP} \xleftarrow{\$} \mathsf{T.Setup}(1^k)$:

   $$\mathsf{bp} = (p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g, \hat{g}) \leftarrow \mathsf{GrGen}(1^k)$$

   All n issuers engage in Gennaro et al.'s distributed key generation protocol (Fig. 5).

   $$(\{\hat{\mathsf{ivk}}_j\}_{j=1}^n, \{\mathsf{isk}_j\}_{j=1}^n) \xleftarrow{\$} \mathsf{New.DKG}(t, n)$$

   where $\{\hat{\mathsf{ivk}}_j\}_{j=1}^n = \{\hat{g}^{\mathsf{isk}_j}\}_{j=1}^n$.
   The system's public parameters are as follows:

   $$\mathsf{PP} \leftarrow (\mathsf{bp}, \{\hat{\mathsf{ivk}}_j\}_{j=1}^n, W, \hat{W})$$

   where $W \xleftarrow{\$} \mathbb{G}$, and $\hat{W} \xleftarrow{\$} \hat{\mathbb{G}}$ and the discrete logarithms base g and $\hat{g}$, respectively, are not known to anyone. This can be achieved with Steps 1-3 in Fig. 5. PP is publicly announced to all network entities.
2. $(\mathsf{usk}, \hat{\mathsf{usk}}) \leftarrow \mathsf{T.Issue}(s, \{\mathsf{isk}_j\}_{j \in \mathcal{S}})$:
   T.Issue is an interactive protocol between a prospective user P offering a real-world unique identifier s and a set of at least t issuers holding $\{\mathsf{isk}_j\}_{j \in \mathcal{S}}$ where $\mathcal{S} \subseteq [n]$ and $|\mathcal{S}| \geq t$. It produces the user's secret key pair $(\mathsf{usk}, \hat{\mathsf{usk}})$.
   At a high level, the protocol is as follows:
   - The user P verifiably secret shares s among the issuers.
   - Each issuer who holds a share of s engages with other issuers in a secure multi-party computation (MPC) protocol to compute DY-VRF† shares (Fig. 4), which are shares of user secret keys.
   - The user reconstructs its secret key pair $(\mathsf{usk}, \hat{\mathsf{usk}})$ using the shares received from the issuers.

   In more detail, the steps are as follows:
   (a) The user P executes the Pedersen-VSS protocol to share s with the issuers and proves that $R(\mathtt{x}_s, (s, \mathtt{w}_s))$ holds. For example, in addition to the Pedersen-VSS, one way to accomplish this is to first (1) Shamir secret share s to obtain $s_1, \ldots, s_n$, (2) compute a Pedersen commitment for each share s_i using randomness r_i, and (3) provide a proof of knowledge of $(s, \mathtt{w}_s)$ together with $\{s_i\}_{i=1}^n, \{r_i\}_{i=1}^n, \{\mathsf{coe}_i\}_{i=1}^{t-1}$, where coe_i is the i^{th} coefficient of the associated polynomial of Shamir secret sharing. P broadcasts all commitments to shares via standard Byzantine broadcast [GKKZ11] to all issuers and, at the same time, securely sends each issuer's share and randomness of the Pedersen commitment to them. Upon receiving messages from a secure broadcast channel, each issuer can check the well-formedness of its commitment with respect to the set of all commitments received from the broadcast channel.

(b) Each issuer Iss_i (who holds s_i from the output of the Pedersen-VSS protocol) engages in a distributed computation of the user's secret key pair as follows:

i. Compute
$$\mu_i = \mathsf{isk}_i + s_i$$

ii. We assume that before engaging in the $\mathsf{T.Issue}$ protocol, issuers have obtained secret shares of a random Beaver triple $[a]$, $[b]$ and $[c] = [a \cdot b]$ using the protocol in [KPR18]. Hence, Iss_i has a_i, b_i and c_i. Holding ρ_i (a secret share of a random field element ρ, which can be generated with Steps 1-3 in Fig. 5), distribute $\rho_i + a_i$ and $\mu_i + b_i$ so that the values of $\rho + a = X$ and $\mu + b = Y$ are revealed publicly.

iii. Compute the secret sharing of $\rho\mu$ as follows:
$$\begin{aligned}\rho\mu &= ((\rho + a) - a)((\mu + b) - b) \\ &= (\rho + a)(\mu + b) - a(\mu + b) - b(\rho + a) + ab \\ &= XY - aY - bX + ab\end{aligned}$$
so that:
$$(\rho\mu)_j = XY - a_jY - b_jX + c_j$$
Distribute $(\rho\mu)_i$ so that the value $(\rho\mu)^{-1}$ can be computed by anyone. Then, the share of the inverse of the field element is computed as follows:
$$1/\mu_i = (\rho\mu)^{-1}\rho_i$$

iv. Compute
$$\mathsf{usk}_i = g^{1/\mu_i}, \quad \hat{\mathsf{usk}}_i = \hat{g}^{1/\mu_i}$$
which are sent to the user via calling $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SRA}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{P}, (\mathsf{usk}_i, \hat{\mathsf{usk}}_i))$.

(c) The user does the following:

i. Receive $(\mathtt{Received}, \mathsf{sid}, \mathsf{Iss}_i, (\mathsf{usk}_i, \hat{\mathsf{usk}}_i))$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SRA}}$.

ii. Reconstruct the secret key pair based on the received shares using Lagrange interpolation (where $\mu = \mathsf{isk} + s$):
$$\mathsf{usk} = g^{1/(s+\mathsf{isk})}, \quad \hat{\mathsf{usk}} = \hat{g}^{1/(s+\mathsf{isk})}$$

Check if $e(\mathsf{usk}, \hat{g}) = e(g, \hat{\mathsf{usk}})$ and $e(\mathsf{usk}, \hat{\mathsf{ivk}} \cdot \hat{g}^s) = e(g, \hat{g})$ hold.

Signature Generation, and *Verification* are the same as in Section 4.

7 Deployment Considerations and Applications

In this section we discuss deployment considerations and applications for SyRA signatures. In particular, we describe how recovery of keys can be handled, how the personhood relation may be realized in an actual deployment, how we can combine revocation and attribute based credentials with SyRA, and how SyRA

New.DKG(t, n)

Generating x:

1. Each party P_i performs a Pedersen-VSS of a random value z_i as a dealer:
(a) P_i chooses two random polynomials $f_i(z), f'_i(z)$ over $\mathbb{Z}_p$ of degree $t-1$:

$$f_i(z) = a_{i,0} + a_{i,1}z + \cdots + a_{i,t-1}z^{t-1} \qquad f'_i(z) = b_{i,0} + b_{i,1}z + \cdots + b_{i,t-1}z^{t-1}$$

Let $z_i = a_{i,0} = f_i(0)$. P_i broadcasts $C_{i,k} = g^{a_{i,k}}h^{b_{i,k}} \pmod p$ for $k = 0, \ldots, t-1$. P_i computes the secret shares $\bar{s}_{i,j} = f_i(j), \bar{s}'_{i,j} = f'_i(j) \pmod p$ for $j = 1, \ldots, n$ and sends $\bar{s}_{i,j}, \bar{s}'_{i,j}$ to party P_j.
(b) Each party P_j verifies the shares they received from the other parties. For each $i = 1, \ldots, n$, P_j checks if

$$g^{\bar{s}_{i,j}}h^{\bar{s}'_{i,j}} = \prod_{k=0}^{t-1} C_{i,k}^{j^k} \pmod p \tag{1}$$

If the check fails for an index i, P_j broadcasts a *complaint* against P_i.
(c) Each party P_i who, as a dealer, received a complaint from party P_j broadcasts the values $\bar{s}_{i,j}, \bar{s}'_{i,j}$ that satisfy Eq. (1).
(d) Each party marks as *disqualified* any party that either
- received more than $t-1$ complaints in Step 1(b), or
- answered a complaint in Step 1(c) with values that falsify Eq. (1).

2. Each party then builds the set of non-disqualified parties $QUAL$.
3. The distributed secret value x is not explicitly computed by any party, but it equals $x = \sum_{i \in QUAL} z_i \pmod p$. Each party P_i sets her share of the secret as $x_i = \sum_{j \in QUAL} \bar{s}_{j,i} \pmod p$ and the value $x'_i = \sum_{j \in QUAL} \bar{s}'_{j,i} \pmod p$.

Extracting $Y = g^x \pmod p$:

4. Each party $i \in QUAL$ exposes $Y_i = g^{z_i} \pmod p$ via Feldman-VSS:
(a) Each party $P_i, i \in QUAL$, broadcasts $A_{i,k} = g^{a_{i,k}} \pmod p$ for $k = 0, \ldots, t-1$.
(b) Each party P_j verifies the values broadcast by the other parties in $QUAL$, namely, for each $i \in QUAL$, P_j checks if

$$g^{\bar{s}_{i,j}} = \prod_{k=0}^{t-1} (A_{i,k})^{j^k} \pmod p \tag{2}$$

If the check fails for an index i, P_j *complains* against P_i by broadcasting the values $\bar{s}_{i,j}, \bar{s}'_{i,j}$ that satisfy Eq. (1) but do not satisfy Eq. (2). 5. For parties P_i who receive at least one valid complaint, i.e., values which satisfy Eq. (1) and not Eq. (2), the other parties run the reconstruction phase of Pedersen-VSS to compute $z_i, f_i(z), A_{i,k}$ for $k = 0, \ldots, t$ in the clear. For all parties in $QUAL$, set $Y_i = A_{i,0} = g^{z_i} \pmod p$. Compute $Y = \prod_{i \in QUAL} Y_i \pmod p$.

Fig. 5. Gennaro et al.'s distributed key generation protocol [GJKR99].

signatures can be used to solve various use cases, such as e-voting, cryptocurrency airdrops and peer DIDs.

Key Recovery. Sybil resilience may complicate key recovery since a Sybil attacker can pretend that it has lost her key in order to be reissued a credential and in this way break Sybil resilience. Various approaches have been suggested to mitigate this attack strategy in prior work, e.g., the key can be revoked using a credential revocation mechanism, e.g., [BCD+17], and only then the user is reissued its new credential. Alternatively the system can store the key information separately in a distributed fashion to facilitate key recovery directly, cf. [MMZ+21]. Our construction approach offers a strikingly simpler key recovery mechanism: A user simply reruns the issuing protocol with the issuer to recover their key, which is always the same for a given identity s. Hence, the system prevents a malicious user who attempts to reissue a DID as part of an attack that evades the system's abuse mitigation mechanisms.

Realizing Personhood. One way to provide proof of personhood in our system is to use an oracle system like DECO [ZMM+20] (or Town Crier [ZCC+16]) that ensures data privacy and remains compatible with legacy systems without requiring alterations to data sources. DECO functions as a three-party protocol involving a prover, a verifier, and a TLS server. Its purpose is to enable the prover to convince the verifier that a piece of data, which may potentially be private to the prover, retrieved from the server satisfies a specific predicate using zero-knowledge proofs without server-side modifications. In our specific context, the DECO verifier can either be the issuer itself or a party designated as trustworthy by the issuer. Using this approach for personhood would require identifying a piece of information that is unique to individuals and possible to verify against a web service. For instance, a social security number (SSN) and a government web-site that includes the SSN as part of a TLS encrypted interaction. Another approach to establishing personhood is using biometric data — a recent embodiment of this idea that gained some traction is Worldcoin [Wor23]. It utilizes a biometric device that reads the iris of an individual and records information that can be utilized to subsequently prove the user's identity.

Revocation. Incorporating revocation comprehensively with SyRA signatures is out of scope in the current writeup and an interesting direction for future work. Nevertheless, we outline here an initial set of ideas to illustrate how it is possible to add credential revocation in different scenarios. A simple method to incorporate revocation is by exploiting the secret key component $\hat{\mathsf{usk}}$ and utilize it in the context of a revocation list. Specifically, a revocation authority can maintain a list of $\hat{\mathsf{usk}}$ values associated with the identity strings and disclose publicly those that need to form the revocation list; alternatively, the issuer can reconstruct $\hat{\mathsf{usk}}$ on the spot given the identity string s of the user that should be revoked. Using this list, a verifier can locally compute $T_{\mathsf{rev}} = e(\mathsf{h}(\mathsf{ctx}), \hat{\mathsf{usk}})$ for each announced $\hat{\mathsf{usk}}$. Given this, the verifier can ignore all signatures with T' values such that $T' = T_{\mathsf{rev}}$. Note that despite disclosing $\hat{\mathsf{usk}}$, no one can sign on behalf of the associated user, as the signature $\sigma = (\mathtt{x}, \pi)$ is a NIZK proof whose generation necessarily requires a witness for the relation $R(\mathtt{x}, \mathtt{w}) = 1$ from

Section 4. This means signature generation also requires usk, which cannot be derived from $\hat{\mathsf{usk}}$ because SyRA relies on Type-III (asymmetric) bilinear groups. In this way, by efficiently revealing only one group element, $\hat{\mathsf{usk}}$, it is possible for verifiers identify all signatures of the revoked user upon receiving their signatures.

The above mechanism requires revealing $\hat{\mathsf{usk}}$. In case the user's SyRA key is compromised, the user can provide s to the issuer along with the relevant proof of ownership of s; so that the issuer computes $\hat{\mathsf{usk}}$. In this setting, the affected user would have to update their identity string s with a new one from the identity provider that is acknowledged by the personhood relation R (e.g., get issued a new social security number). Note that the identity provider needs to check that the old identity string has indeed been revoked. Furthermore, this will not provide forward security, at least in a non-interactive manner since revealing the $\hat{\mathsf{usk}}$ value of a user will link all its past pseudonyms. (Note that alternatively, verifiers could, in principle, run a multi-party protocol with issuers to check if an identity has been revoked instead of using $\hat{\mathsf{usk}}$ but this is not a practical solution). Adding forward security to this revocation mechanism is an interesting question for future work.

Identity Attributes. SyRA signatures can be combined with anonymous attribute based credentials to demonstrate user attributes in a privacy-preserving way while maintaining resilience against Sybil attacks. The approach is simple: users will obtain keys for both SyRA and anonymous attribute-based credentials. This allows subsequently to engage in attribute-based identification with each transcript signed using SyRA for the context requested by the verifier. While simple and effective the above approach may be further improved in terms of efficiency by investigating a concrete design for attribute-based SyRA signatures; we leave this question for future work.

7.1 Applications

e-Voting and e-Petitions. There are many online services that require strict single access per ID, such as e-voting. SyRA enables verifiers to cluster all signatures with respect to their pseudonyms T for each context. Sybil resilience means that a user can create at most one pseudonym T per context. For instance, in e-voting or e-petitions [DKD+08], the election procedure identifier would be used to determine the context, and the choice of the voter will be the message. The user can sign multiple times for the same identifier (as is needed in many election schemes for coercion protection, for instance, [VG06]), and then the system can choose the latest "message" (recall all signatures of the user share the same T value).

Cryptocurrency Airdrops. In a cryptocurrency airdrop, a user is set to receive a certain amount of cryptocurrency. The user generates a cryptocurrency address A; utilizing SyRA signatures, the users can use the name of the airdrop event as the context and the address A as the message to be signed. The primitive

enables the user to sign multiple addresses if needed or perform various actions in the context of one specific airdrop. Note that if a user signs the same context twice, those two signatures are linkable (they have the same T), however still, the privacy of the user's real-world identifier, s, is preserved and the user cannot be linked if they participate in multiple airdrops. Our mechanism enables the airdrop provider to write policies such as "each user receives exactly X coins", with the Sybil resilient property of SyRA being the enforcer of this property. [10]

Peer DID. Decentralized identifiers are often anchored to a public source of truth, such as a blockchain. While this ensures the security and immutability of transactions, it allows arbitrary parties to resolve users' DIDs (i.e., obtain their DID documents). Users in a DID system should control every aspect of the use of their digital identity, including the ability to interact with other people, institutions, and organizations in a private manner. One solution, put forth by the Decentralized Identity Foundation [Fou23], is Peer DID[11], which moves the resolution of DIDs to local, peer-to-peer interactions off chain. In this system, a user Alice can create a pairwise DID for each interaction with a different user or organization. For example, Alice may interact with her doctor, bank, etc. using these context-specific "credentials." Our SyRA signatures are directly applicable here, as Alice can create pairwise DIDs from her master credential, which is tied to her real-world identity, via context-specific pseudonyms. The security properties of SyRA signatures ensure that Alice's identity is protected, and her interactions are unlinkable across all contexts in which she engages, while Sybil resilience protects the verifiers from interacting with multiple users controlled by Alice.

Privacy-Preserving Regulatory Compliance. In the context of regulatory compliance, SyRA signatures provide a mechanism for verifying identities without exposing sensitive information. This capability is particularly relevant for financial institutions and other regulated entities that need to adhere to KYC (Know Your Customer), AML (Anti Money Laundering), and CFT (Combating Financing Terrorism) regulations. For example, when a recipient is involved in a transaction, they may be asked to sign a transaction using SyRA to verify that they are not identified as a terrorist according to a list of terrorist identity strings. Such a list could be prepared by the issuer for a specific context of interest by calculating all the relevant T-values; note that this also maintains the privacy of the list itself. Adding this requirement to the user's workflow would ensure compliance with CFT regulations while enabling the recipient to receive funds without disclosing her identity. Similarly, the sender of a transaction can be required to provide a SyRA signature on the transaction to prove that she is not associated with any revoked entities listed in AML regulations,

[10] As an example of the relevance of our approach to practice consider that in the Arbitrum airdrop, around 253 million Arbitrum tokens, 21.8% of all tokens, were acquired by Sybil accounts [Xe23].

[11] https://identity.foundation/peer-did-method-spec/index.html

enabling her to send funds. SyRA signatures can also be used to facilitate AML/CFT checks in anonymous payment systems like [GGM16], PEReDi [KKS22], Zether [BAZB20], and PARScoin [SKK23], following the mechanism described above offering a privacy-preserving, yet compliant, solution for these platforms.

Acknowledgements

We would like to express our gratitude to Markulf Kohlweiss for the valuable discussions and suggestions provided. This work has been supported by Input Output (iohk.io) through their funding of the University of Edinburgh Blockchain Technology Lab.

References

ALSY13. Man Ho Au, Joseph K. Liu, Willy Susilo, and Tsz Hon Yuen. Secure id-based linkable and revocable-iff-linked ring signature with constant-size construction. *Theor. Comput. Sci.*, 469:1–14, 2013.

BAZB20. Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In *International Conference on Financial Cryptography and Data Security*, pages 423–443. Springer, 2020.

BCD+17. Foteini Baldimtsi, Jan Camenisch, Maria Dubovitskaya, Anna Lysyanskaya, Leonid Reyzin, Kai Samelin, and Sophia Yakoubov. Accumulators with applications to anonymity-preserving revocation. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*, pages 301–315. IEEE, 2017.

BCJ+24. Foteini Baldimtsi, Konstantinos Kryptos Chalkias, Yan Ji, Jonas Lindstrøm, Deepak Maram, Ben Riva, Arnab Roy, Mahdi Sedaghat, and Joy Wang. zklogin: Privacy-preserving blockchain authentication with existing credentials. *CoRR*, abs/2401.11735, 2024.

Bea92. Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Heidelberg, Germany.

BLS04. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, September 2004.

CDG+18. Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 280–312. Springer, 2018.

CE86. David Chaum and Jan-Hendrik Evertse. A secure and privacy-protecting protocol for transmitting personal information between organizations. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 118–167. Springer, 1986.

Cha81. David Chaum. Verification by anonymous monitors. In Allen Gersho, editor, *Advances in Cryptology – CRYPTO'81*, volume ECE Report 82-04, pages 138–139, Santa Barbara, CA, USA, 1981. U.C. Santa Barbara, Dept. of Elec. and Computer Eng.

Cha85. David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, 1985.

CHK+06. Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006*, pages 201–210. ACM, 2006.

CHL05. Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 302–321, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany.

CHL06. Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Balancing accountability and privacy using e-cash (extended abstract). In Roberto De Prisco and Moti Yung, editors, *SCN 06: 5th International Conference on Security in Communication Networks*, volume 4116 of *Lecture Notes in Computer Science*, pages 141–155, Maiori, Italy, September 6–8, 2006. Springer, Heidelberg, Germany.

CL01. Jan Camenisch and Anna Lysyanskaya. An efficient system for nontransferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118, Innsbruck, Austria, May 6–10, 2001. Springer, Heidelberg, Germany.

Con22. The World Wide Web Consortium. Decentralized identifiers (dids), 2022. https://w3c.github.io/did-core/.

Cv91. David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *Advances in Cryptology – EUROCRYPT'91*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265, Brighton, UK, April 8–11, 1991. Springer, Heidelberg, Germany.

DDP06. Ivan Damgård, Kasper Dupont, and Michael Østergaard Pedersen. Unclonable group identification. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 555–572. Springer, 2006.

DKD+08. Claudia Diaz, Eleni Kosta, Hannelore Dekeyser, Markulf Kohlweiss, and Girma Nigusse. Privacy preserving electronic petitions. *Identity in the Information Society*, 1(1):203–219, 2008.

DMS04. Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In Matt Blaze, editor, *USENIX Security 2004: 13th USENIX Security Symposium*, pages 303–320, San Diego, CA, USA, August 9–13, 2004. USENIX Association.

DY05. Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *International Workshop on Public Key Cryptography*, pages 416–431. Springer, 2005.

EG14. Alex Escala and Jens Groth. Fine-tuning Groth-Sahai proofs. In Hugo Krawczyk, editor, *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 630–649, Buenos Aires, Argentina, March 26–28, 2014. Springer, Heidelberg, Germany.

ElG84. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18, Santa Barbara, CA, USA, August 19–23, 1984. Springer, Heidelberg, Germany.

FKS16. Daniel Fett, Ralf Küsters, and Guido Schmitz. A comprehensive formal security analysis of oauth 2.0. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1204–1215, New York, NY, USA, 2016. Association for Computing Machinery.

Fou23. Decentralized Identity Foundation, 2023. https://identity.foundation/.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.

FS07. Eiichiro Fujisaki and Koutarou Suzuki. Traceable ring signature. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007: 10th International Conference on Theory and Practice of Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 181–200, Beijing, China, April 16–20, 2007. Springer, Heidelberg, Germany.

GGM16. Christina Garman, Matthew Green, and Ian Miers. Accountable privacy for decentralized anonymous payments. In Jens Grossklags and Bart Preneel, editors, *FC 2016: 20th International Conference on Financial Cryptography and Data Security*, volume 9603 of *Lecture Notes in Computer Science*, pages 81–98, Christ Church, Barbados, February 22–26, 2016. Springer, Heidelberg, Germany.

GJKR99. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 295–310, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany.

GJM+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, *Advances in Cryptology – EUROCRYPT 2021, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 147–176, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany.

GKKZ11. Juan A Garay, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. Adaptively secure broadcast, revisited. In *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 179–186, 2011.

GOS06. Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for np. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 339–358. Springer, 2006.

GQ88. Louis C. Guillou and Jean-Jacques Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both trasmission and memory. In C. G. Günther, editor, *Advances in Cryptology – EUROCRYPT'88*, volume 330 of *Lecture Notes in Computer Science*, pages 123–128, Davos, Switzerland, May 25–27, 1988. Springer, Heidelberg, Germany.

Gro10. Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 321–340, Singapore, December 5–9, 2010. Springer, Heidelberg, Germany.

Her06. Javier Herranz. Deterministic identity-based signatures for partial aggregation. *Comput. J.*, 49(3):322–330, 2006.

Hes03. Florian Hess. Efficient identity based signature schemes based on pairings. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002: 9th Annual International Workshop on Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 310–324, St. John's, Newfoundland, Canada, August 15–16, 2003. Springer, Heidelberg, Germany.

JAP09. JAP. Jap anonymity & privacy, 2009. http://anon.inf.tu-dresden.de/.

KKS22. Aggelos Kiayias, Markulf Kohlweiss, and Amirreza Sarencheh. Peredi: Privacy-enhanced, regulated and distributed central bank digital currencies. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1739–1752, 2022.

KPR18. Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 158–189, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

KTY04. Aggelos Kiayias, Yiannis Tsiounis, and Moti Yung. Traceable signatures. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 571–589, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany.

KZ07. Aggelos Kiayias and Hong-Sheng Zhou. Hidden identity-based signatures. In Sven Dietrich and Rachna Dhamija, editors, *FC 2007: 11th International Conference on Financial Cryptography and Data Security*, volume 4886 of *Lecture Notes in Computer Science*, pages 134–147, Scarborough, Trinidad and Tobago, February 12–16, 2007. Springer, Heidelberg, Germany.

LWW04. Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). In Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan, editors, *ACISP 04: 9th Australasian Conference on Information Security and Privacy*, volume 3108 of *Lecture Notes in Computer Science*, pages 325–335, Sydney, NSW, Australia, July 13–15, 2004. Springer, Heidelberg, Germany.

MMZ+21. Deepak Maram, Harjasleen Malvai, Fan Zhang, Nerla Jean-Louis, Alexander Frolov, Tyler Kell, Tyrone Lobban, Christine Moy, Ari Juels, and Andrew Miller. Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1348–1366. IEEE, 2021.

Ped92. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Heidelberg, Germany.

PLL+23. Sanghyeon Park, Jeong Hyuk Lee, Seunghwa Lee, Jung Hyun Chun, Hyeonmyeong Cho, MinGi Kim, Hyun Ki Cho, and Soo-Mook Moon. Beyond the blockchain address: Zero-knowledge address abstraction. *IACR Cryptol. ePrint Arch.*, page 191, 2023.

RST01. Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 552–565, Gold Coast, Australia, December 9–13, 2001. Springer, Heidelberg, Germany.

Sch91. Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991.

Sha79. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

Sha84. Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53, Santa Barbara, CA, USA, August 19–23, 1984. Springer, Heidelberg, Germany.

SKK23. Amirreza Sarencheh, Aggelos Kiayias, and Markulf Kohlweiss. Parscoin: A privacy-preserving, auditable, and regulation-friendly stablecoin. *Cryptology ePrint Archive*, 2023.

SZ21. Alessandra Scafuro and Bihan Zhang. One-time traceable ring signatures. In Elisa Bertino, Haya Shulman, and Michael Waidner, editors, *ESORICS 2021: 26th European Symposium on Research in Computer Security, Part II*, volume 12973 of *Lecture Notes in Computer Science*, pages 481–500, Darmstadt, Germany, October 4–8, 2021. Springer, Heidelberg, Germany.

VG06. Melanie Volkamer and Rüdiger Grimm. Multiple casts in online voting: Analyzing chances. In Robert Krimmer, editor, *Electronic Voting 2006: 2nd International Workshop, Co-organized by Council of Europe, ESF TED, IFIP WG 8.6 and E-Voting.CC, August, 2nd - 4th, 2006 in Castle Hofen, Bregenz, Austria*, volume P-86 of *LNI*, pages 97–106. GI, 2006.

WCW12. Rui Wang, Shuo Chen, and XiaoFeng Wang. Signing me onto your accounts through facebook and google: A traffic-guided security study of commercially deployed single-sign-on web services. In *2012 IEEE Symposium on Security and Privacy*, pages 365–379, 2012.

Wor23. Worldcoin, 2023. https://whitepaper.worldcoin.org/.

Xe23. X-explore. Advanced analysis for arbitrum airdrop, 2023. https://mirror.xyz/x-explore.eth/AFroG11e24I6S1oDvTitNdQSDh8lN5bz9VZAink8lZ4.

ZCC+16. Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town crier: An authenticated data feed for smart contracts. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 270–282, Vienna, Austria, October 24–28, 2016. ACM Press.

ZMM+20. Fan Zhang, Deepak Maram, Harjasleen Malvai, Steven Goldfeder, and Ari Juels. DECO: Liberating web data using decentralized oracles for TLS. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1919–1938, Virtual Event, USA, November 9–13, 2020. ACM Press.

A Additional Definitions

Definition 3 (ElGamal Encryption Scheme). [ElG84] *The ElGamal encryption scheme consists of the following three PPT algorithms:*

1. *Key Generation. Assuming that p is a large prime and g is a generator of the group $\mathbb{Z}_p^*$, the key generation algorithm is performed as follows:*
   - *Randomly select a secret key $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p^*$.*
   - *Compute the corresponding public key as $\mathsf{pk} = g^{\mathsf{sk}}$.*
   - *Make the parameters (g, p, pk) publicly available.*
2. *Encryption. To encrypt a message $m \in \mathbb{Z}_p$, the encryption algorithm is performed as follows:*
   - *Randomly select $k \xleftarrow{\$} \mathbb{Z}_p^*$.*
   - *The ciphertext $\chi = (C_1, C_2) \mod p$ is computed as $C_1 = g^k$, and $C_2 = \mathsf{pk}^k \cdot m$.*
3. *Decryption. Given a ciphertext $\chi = (C_1, C_2)$, the decryption algorithm is performed as follows:*
   - *Compute the message m as: $m = C_2 / C_1^{\mathsf{sk}}$.*

The ElGamal encryption scheme is IND-CPA secure under the decisional Diffie-Hellman (DDH) assumption [ElG84].

Definition 4 (CPA Security of Public Key Encryption Scheme). *Let $\mathsf{PE} = (\mathsf{PE.KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a public key encryption scheme. The following security experiment is conducted between a PPT adversary $\mathcal{A}$ and a challenger. This security experiment $\mathsf{IND\text{-}CPA}^b_{\mathsf{PE}}(\mathcal{A}, k)$ is parameterized by a bit $b \in \{0, 1\}$ and a security parameter k.*

1. *The challenger randomly selects a key pair $(\mathsf{pk}, \mathsf{sk})$ by executing $\mathsf{PE.KeyGen}(1^k)$ and provides the public key pk to the adversary $\mathcal{A}$.*
2. *The adversary $\mathcal{A}$ submits two plaintext messages, denoted as (m_0, m_1), where $|m_0| = |m_1|$.*
3. *The challenger encrypts one of the plaintexts based on the bit b to create a ciphertext $c_b = \mathsf{Enc}_{\mathsf{pk}}(m_b)$ and provides this ciphertext c_b to the adversary $\mathcal{A}$.*
4. *The adversary $\mathcal{A}$ outputs a bit b', indicating its guess of the value of b. If $\mathcal{A}$ decides to abort without producing an output, the output bit b' is set to 0.*

The adversary's advantage is defined as:

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}}(k) = |\Pr[\mathsf{IND\text{-}CPA}^1_{\mathsf{PE}}(\mathcal{A}, k) = 1] - \Pr[\mathsf{IND\text{-}CPA}^0_{\mathsf{PE}}(\mathcal{A}, k) = 1]|$$

The public key encryption scheme PE is $\mathsf{IND\text{-}CPA}$ secure if for all PPT adversaries $\mathcal{A}$ in this experiment, there exists a negligible function ν such that:

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}}(k) \leq \nu(k)$$

Definition 5 (Random Oracle). *$\mathcal{F}_{\mathsf{RO}}$ models an idealized hash function [CDG+18] and is defined as follows.*

Functionality $\mathcal{F}_{\mathsf{RO}}$

The functionality is parameterized by a message space M and output space H and acts as follows: Upon receiving $(\mathtt{Query}, \mathsf{sid}, m)$ from a party P: (i) Ignore if $m \notin M$. (ii) If there exists a tuple (sid, m', h') where $m' = m$, set $h \leftarrow h'$. (iii) Else, select $\bar{h} \xleftarrow{\$} H$ such that there is no stored tuple (sid, m^*, h') where $h' = \bar{h}$. Set $h \leftarrow \bar{h}$. (iv) Store (sid, m, h). (v) Output $(\mathtt{Query.Re}, \mathsf{sid}, h)$ to party P.

Definition 6 (Non-Interactive Zero-Knowledge). *In Section 4.1, we discussed the implementation of non-interactive zero-knowledge proofs (NIZKs) used in our construction. Groth et al. [GOS06] provided a formal definition of non-interactive zero knowledge via the following ideal functionality $\mathcal{F}_{\mathsf{NIZK}}$.*

Functionality $\mathcal{F}_{\mathsf{NIZK}}$

$\mathcal{F}_{\mathsf{NIZK}}$ is parameterized by a relation R.
Proof Generation: *(i) On receiving $(\mathtt{Prove}, \mathsf{sid}, \mathtt{x}, \mathtt{w})$ from some party P, ignore if $R(\mathtt{x}, \mathtt{w}) = 0$. Else, send $(\mathtt{Prove}, \mathsf{sid}, \mathtt{x})$ to $\mathcal{A}$. (ii) Upon receiving $(\mathtt{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{A}$, store $(\mathtt{x}, \pi)$ and send $(\mathtt{Proof}, \mathsf{sid}, \pi)$ to P.*
Proof Verification: *(i) Upon receiving $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$ from some party P, check whether $(\mathtt{x}, \pi)$ is stored. If not, send $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$ to $\mathcal{A}$. (ii) Upon receiving response $(\mathtt{Witness}, \mathsf{sid}, \mathtt{w})$ from $\mathcal{A}$, if $R(\mathtt{x}, \mathtt{w}) = 1$ store $(\mathtt{x}, \pi)$. If $(\mathtt{x}, \pi)$ has been stored, output $(\mathtt{Verification}, \mathsf{sid}, 1)$ to P; else, output $(\mathtt{Verification}, \mathsf{sid}, 0)$.*

Definition 7 (Communication Channel). *Our functionality $\mathcal{F}_{\mathsf{SyRA}}$ does not reveal the identities of users. To realize this functionality, our protocol employs different kinds of communication channels $\mathcal{F}_{\mathsf{Ch}}$ to deliver messages and to fulfill network-level anonymity.*

As discussed in Section 4, regardless of the robustness of cryptographic safeguards at the protocol level, "network leakage" could potentially expose the parties involved in a communication. Hence, in our formal UC modelling, a degree of sender/receiver anonymity is essential for maintaining (UC) anonymity. UC formalism aside, we presume that the user/issuer directs its communication through an underlying anonymous communication infrastructure, like common networks such as Tor [DMS04] or JAP [JAP09], to avoid being identified by IP addresses (using anonymous communication networks is crucial for any solution relying on anonymity for privacy protection).

Functionality $\mathcal{F}_{\mathsf{Ch}}$

Let $\mathtt{S}$ *be the sender and* $\mathtt{R}$ *the receiver of a message* m*.*

1. *Upon input* $(\mathtt{Send}, \mathsf{sid}, \mathtt{R}, m)$ *from* $\mathtt{S}$*, record a mapping* $\mathsf{P}(\mathsf{mid}) \leftarrow (\mathtt{S}, \mathtt{R}, m)$ *where* mid *is chosen at random. Output* $(\mathtt{Send}, \mathsf{sid}, \alpha, \beta, \mathsf{mid})$ *to* $\mathcal{A}$*.*
2. *Upon receiving* $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid})$ *from* $\mathcal{A}$*, retrieve* $(\mathtt{S}, \mathtt{R}, m) \leftarrow \mathsf{P}(\mathsf{mid})$ *and send* $(\mathtt{Received}, \mathsf{sid}, \mathtt{S}, m)$ *to* $\mathtt{R}$*.*

– *Once* $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SSA}}$ *is called, set* $\alpha \leftarrow \mathtt{R}, \beta \leftarrow |m|$ *(secure and sender-anonymous channel).*
– *Once* $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{SRA}}$ *is called, set* $\alpha \leftarrow \mathtt{S}, \beta \leftarrow |m|$ *(secure and receiver-anonymous channel).*