Delegatable Anonymous Credentials from Mercurial Signatures

Elizabeth C. Crites* and Anna Lysyanskaya**

Brown University
Providence, RI 02912

September 27, 2018

Abstract. In a delegatable anonymous credential system, participants may use their credentials anonymously as well as anonymously delegate them to other participants. Such systems are more usable than traditional anonymous credential systems because a popular credential issuer can delegate some of its responsibilities without compromising users' privacy. They also provide stronger privacy guarantees than traditional anonymous credential systems because the identities of credential issuers are hidden. The identity of a credential issuer may convey information about a user's identity even when all other information about the user is concealed.

The only previously known constructions of delegatable anonymous credentials were prohibitively inefficient. They were based on non-interac tive zero-knowledge (NIZK) proofs. In this paper, we provide a simple construction of delegatable anonymous credentials and prove its security in the generic group model. Our construction is direct, not based on NIZK proofs, and is therefore considerably more efficient. In fact, in our construction, only five group elements are needed per link to represent an anonymous credential chain.

Our main building block is a new type of signature scheme, a *mercurial* signature, which allows a signature σ on a message M under public key pk to be transformed into a signature σ' on an equivalent but unlinkable message M' under an equivalent but unlinkable public key pk'.
Keywords: Anonymous credentials, signature schemes, generic group model.

1 Introduction

Anonymous credentials. Anonymous credentials allow a user to prove possession of a set of credentials, issued by some trusted issuer or issuers, that allow access to a resource. What makes them *anonymous* is the fact that the user's proof is zero-knowledge and credentials can be obtained anonymously: an issuer need not know the user's identity in order to issue a credential.

* Email: elizabeth_crites@brown.edu. This work was supported by NSF grant 1422361.

** Email: anna_lysyanskaya@brown.edu. URL: https://cs.brown.edu/people/anna/. This work is supported by NSF grant 1422361.

As a result of decades of research, there are anonymous credential systems that are provably secure and efficient enough for practical use [Cha86,LRSW99] [CL01,Lys02,CL04,CKL+14,CDHK15]. These results have attracted wide attention beyond the cryptographic community: they have been implemented by industry leaders such as IBM, incorporated into industrial standards (such as the TCG standard), and underpinned government policy.

And yet traditional anonymous credentials *do not in fact protect users' privacy.* The traditional anonymous credential model assumes that the verifying party, such as an access provider, knows the public key of the credential issuer or issuers. This can reveal a lot of information about a user. In the US, for example, the identity of the issuer of a user's driver's license (the local DMV) might reveal the user's zip code. If, in addition, the user's date of birth and gender are leaked (as could happen in the context of a medical application), this is enough to uniquely identify the user the majority of the time [Swe97].

A naive approach to remedy this is as follows. Instead of proving possession of a credential from a particular issuer (a particular DMV), a user proves possession of a credential from *one* issuer out of a long list. This simple solution is undesirable for two reasons: (1) it incurs a significant slowdown, proportional to the number of potential issuers, and (2) it requires the user herself to know who the issuer is.

Delegatable anonymous credentials. A more promising approach is to use delegatable anonymous credentials [CL06] [BCC+09]. First, a non-anonymous delegatable credential scheme can be constructed as follows. A certification chain is rooted at some authority and ends at the public key of the user in question, who then needs to demonstrate that she knows the corresponding secret key to prove that she is authorized. The simplest case, when the trusted authority issues certificates *directly* to each user (so the length of each certification chain is 1), is inconvenient because it requires the authority to do too much work. A system in which the authority delegates responsibility to other entities is more convenient: an entity with a certification chain of length ℓ can issue certification chains of length $\ell + 1$. A conventional signature scheme immediately allows delegatable credentials: Alice, who has a public signing key pk_A and a certification chain of length ℓ, can sign Bob's public key pk_B, which gives Bob a certification chain of length $\ell + 1$.

Delegatable anonymous credentials allow users to enjoy much more privacy. Even the users themselves do not know the actual identities of the links on their certification chains. They only know what they need to know. For example, consider a discount program for senior citizens. An online shopper proves that she is eligible for the discount by presenting a level-3 credential from the government administering the program as follows. Government official Alice receives a credential directly from the government. She gives a credential to a local grocer, Bob, who does not need to know who she is or to whom she has issued credentials. Bob's job is to issue credentials to his customers who are senior citizens and he gives such a credential to Carol. Carol need not know who Bob is, who gave him the credential, or who else received credentials from him. Now Carol

can use her credential to shop online with a discount. Her credential does not reveal the identity of anyone on her credential chain. Thus, even if Bob issues a discount credential to no other customer, Carol's anonymity is still preserved.

Delegatable anonymous credentials (DACs) were first proposed by Chase and Lysyanskaya [CL06], who gave a proof of concept construction based on non-interactive zero-knowledge (NIZK) proof systems for NP. Their construction incurred a blow-up that was exponential in L, the length of the certification chain. Even for constant L it was not meant for use in practice. Belenkiy et al. [BCC+09] showed that, given a commitment scheme and a signature scheme that "play nicely" with randomizable NIZK (which they defined and realized), DACs with only linear dependency on L could be achieved. They also showed that their approach could be instantiated using Groth-Sahai commitments and an NIZK proof system [GS08]. Although this was a significant efficiency improvement over previous work, the resulting scheme's use of heavy machinery, such as the Groth-Sahai proof system, rendered it unsuitable for use in practice (a back-of-the-envelope calculation shows that several hundred group elements would be required to represent a certification chain of length two.) Chase et al. [CKLM13] gave a conceptually novel construction of delegatable anonymous credentials that relied on controlled-malleable signatures and achieved stronger security; however, their instantiation of controlled-malleable signatures still required Groth-Sahai proofs, so the resulting construction was essentially as inefficient as that of Belenkiy et al. A recent paper by Camenisch et al. [CDD17] suggests a solution in which one can indeed prove possession of a credential chain in a privacy-preserving manner, but one cannot obtain credentials anonymously.

Our contribution. We provide a simple and efficient construction of delegatable anonymous credentials. Our construction does not rely on heavy machinery such as NIZK proofs. It relies on groups with bilinear pairings, and only five group elements per level of delegation are needed to represent a credential chain (i.e. if Alice obtains a credential from the certification authority (CA) and delegates it to Bob, who in turn delegates it to Carol, Carol's credential chain can be represented using fifteen group elements.) Our construction is provably secure in the generic group model. We also give what we believe to be a simpler definition of DACs.

Our approach. The main building block of our construction is a new type of a signature scheme, which we call a *mercurial* signature scheme[1]. Given a mercurial signature σ on a message M under public key pk, one can, without knowing the underlying secret key sk, transform it into a new signature σ' on an equivalent message M' under an equivalent public key pk', for some set of equivalence relations on messages and public keys. Moreover, for an appropriate choice of message space and public key space, this can be done in such a way that the new M' cannot be linked to the original M, and the new public key pk' cannot be linked to pk.

The approach to constructing DACs from mercurial signatures is as follows. Suppose that the certification authority with public key pk_0 has issued a cre-

[1] No relationship to mercurial commitments [CHK+05]

dential to Alice, whose pseudonym is some public key pk_A. Alice's certification chain (of length 1) will have the form $(\mathsf{pk}_A, \sigma_A)$, where σ_A is the CA's signature on pk_A. Alice interacts with Bob, who knows her under a different public key, pk'_A. The public keys pk_A and pk'_A are equivalent — they both belong to Alice and have the same underlying secret key — but Bob cannot link them. Mercurial signatures allow her to translate σ_A into σ'_A, which is the CA's signature on her pk'_A. Alice delegates her credential to Bob, after which Bob's certification chain has the form $((\mathsf{pk}'_A, \mathsf{pk}_B), (\sigma'_A, \sigma_B))$, where pk_B is the pseudonym under which Bob is known to Alice, and σ_B is the signature on pk_B under the public key pk'_A.

Now suppose that Bob wants to delegate to Carol, who is known to him under the pseudonym pk_C. He first uses the properties of the mercurial signature scheme in order to make his credential chain unrecognizable. He transforms pk'_A into an equivalent pk''_A and pk_B into an equivalent pk'_B, taking care to also transform the signatures appropriately. Finally, he signs pk_C under pk'_B.

Our mercurial signatures were inspired by the paper of Fuchsbauer, Hanser and Slamanig [FHS14] on structure-preserving signatures on equivalence classes (SPS-EQ). SPS-EQ does not include the feature of transforming a public key into an equivalent one; this is new with our mercurial signatures. It does introduce the property that a signature on a message M can be transformed into one on an equivalent message M', where M' cannot be linked to M. It also presents a construction of SPS-EQ that is secure in the generic group model. Our construction of mercurial signatures is adapted from theirs, but our notion of security requires that the adapted construction is still unforgeable even when the forger is given the added freedom to modify the public key. In addition, it requires that we prove pk_A and pk'_A are unlinkable even when given signatures under these keys. A recent, independent work by Backes et al. [BHKS18] considers signatures with flexible public keys, but they don't consider a scheme that allows for both messages and public keys to come from an equivalence class.

Open problems. Our paper leaves open the question of how to construct efficient DACs with desirable features that have been explored in the context of anonymous credentials, such as credential attributes (e.g. expiration dates), revocation, identity escrow and conditional anonymity.

2 Definition of Mercurial Signatures

For a relation $\mathcal{R}$ over strings, let $[x]_{\mathcal{R}} = \{y \mid \mathcal{R}(x, y)\}$. If $\mathcal{R}$ is an equivalence relation, then $[x]_{\mathcal{R}}$ denotes the equivalence class of which x is a representative. We say (somewhat loosely) that a relation $\mathcal{R}$ is *parameterized* if it is well-defined as long as some other parameters are well-defined. For example, if $\mathbb{G}$ is a cyclic group with generators g and h, then the decisional Diffie-Hellman (DDH) relation $\mathcal{R} = \{(x, y) \mid \exists \alpha \text{ such that } x = g^\alpha \wedge y = h^\alpha\}$ is parameterized by $\mathbb{G}$, g, and h and is well-defined as long as $\mathbb{G}$, g, and h are well-defined.

Definition 1 (Mercurial signature) *A mercurial signature scheme for parameterized equivalence relations $\mathcal{R}_M$, $\mathcal{R}_{\mathsf{pk}}$, $\mathcal{R}_{\mathsf{sk}}$ is a tuple of the following polynomial-time algorithms, which are deterministic algorithms unless otherwise stated:*

$\mathsf{PPGen}(1^k) \to PP$: On input the security parameter 1^k, this probabilistic algorithm outputs the public parameters PP. This includes parameters for the parameterized equivalence relations $\mathcal{R}_M$, $\mathcal{R}_{\mathsf{pk}}$, $\mathcal{R}_{\mathsf{sk}}$ so they are well-defined. It also includes parameters for the algorithms sample_ρ and sample_μ, which sample key and message converters, respectively.

$\mathsf{KeyGen}(PP, \ell) \to (\mathsf{pk}, \mathsf{sk})$: On input the public parameters PP and a length parameter ℓ, this probabilistic algorithm outputs a key pair $(\mathsf{pk}, \mathsf{sk})$. The message space $\mathcal{M}$ is well-defined from PP and ℓ. This algorithm also defines a correspondence between public and secret keys: we write $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{KeyGen}(PP, \ell)$ if there exists a set of random choices that KeyGen could make that would result in $(\mathsf{pk}, \mathsf{sk})$ as the output.

$\mathsf{Sign}(\mathsf{sk}, M) \to \sigma$: On input the signing key sk and a message $M \in \mathcal{M}$, this probabilistic algorithm outputs a signature σ.

$\mathsf{Verify}(\mathsf{pk}, M, \sigma) \to 0/1$: On input the public key pk, a message $M \in \mathcal{M}$, and a purported signature σ, output 0 or 1.

$\mathsf{ConvertSK}(\mathsf{sk}, \rho) \to \tilde{\mathsf{sk}}$: On input sk and a key converter $\rho \in \mathsf{sample}_\rho$, output a new secret key $\tilde{\mathsf{sk}} \in [\mathsf{sk}]_{\mathcal{R}_{\mathsf{sk}}}$.

$\mathsf{ConvertPK}(\mathsf{pk}, \rho) \to \tilde{\mathsf{pk}}$: On input pk and a key converter $\rho \in \mathsf{sample}_\rho$, output a new public key $\tilde{\mathsf{pk}} \in [\mathsf{pk}]_{\mathcal{R}_{\mathsf{pk}}}$ (correctness of this operation, defined below, will guarantee that if pk corresponds to sk, then $\tilde{\mathsf{pk}}$ corresponds to $\tilde{\mathsf{sk}} = \mathsf{ConvertSK}(\mathsf{sk}, \rho)$.)

$\mathsf{ConvertSig}(\mathsf{pk}, M, \sigma, \rho) \to \tilde{\sigma}$: On input pk, a message $M \in \mathcal{M}$, a signature σ, and key converter $\rho \in \mathsf{sample}_\rho$, this probabilistic algorithm returns a new signature $\tilde{\sigma}$ (correctness of this will require that whenever $\mathsf{Verify}(\mathsf{pk}, M, \sigma) = 1$, it will also be the case that $\mathsf{Verify}(\tilde{\mathsf{pk}}, M, \tilde{\sigma}) = 1$.)

$\mathsf{ChangeRep}(\mathsf{pk}, M, \sigma, \mu) \to (M', \sigma')$: On input pk, a message $M \in \mathcal{M}$, a signature σ, and a message converter $\mu \in \mathsf{sample}_\mu$, this probabilistic algorithm computes a new message $M' \in [M]_{\mathcal{R}_M}$ and a new signature σ' and outputs (M', σ') (correctness of this will require that $\mathsf{Verify}(\mathsf{pk}, M, \sigma) = 1 \Rightarrow \mathsf{Verify}(\mathsf{pk}, M', \sigma') = 1$.)

Similar to a standard cryptographic signature [GMR88], a mercurial signature must be correct and unforgeable.

Definition 2 (Correctness) *A mercurial signature scheme* (PPGen, KeyGen, Sign, Verify, $\mathsf{ConvertSK}$, $\mathsf{ConvertPK}$, $\mathsf{ConvertSig}$, $\mathsf{ChangeRep}$) *for parameterized equivalence relations $\mathcal{R}_M$, $\mathcal{R}_{\mathsf{pk}}$, $\mathcal{R}_{\mathsf{sk}}$ is correct if it satisfies the following conditions for all k, for all $PP \in \mathsf{PPGen}(1^k)$, for all $\ell > 1$, for all $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{KeyGen}(PP, \ell)$:*

Verification For all $M \in \mathcal{M}$, for all $\sigma \in \mathsf{Sign}(\mathsf{sk}, M)$, $\mathsf{Verify}(\mathsf{pk}, M, \sigma) = 1$.

Key conversion For all $\rho \in \mathsf{sample}_\rho$, $(\mathsf{ConvertPK}(\mathsf{pk}, \rho), \mathsf{ConvertSK}(\mathsf{sk}, \rho)) \in \mathsf{KeyGen}(PP, \ell)$. Moreover, $\mathsf{ConvertSK}(\mathsf{sk}, \rho) \in [\mathsf{sk}]_{\mathcal{R}_{\mathsf{sk}}}$ and $\mathsf{ConvertPK}(\mathsf{pk}, \rho) \in [\mathsf{pk}]_{\mathcal{R}_{\mathsf{pk}}}$.

Signature conversion For all $M \in \mathcal{M}$, for all σ such that $\mathsf{Verify}(\mathsf{pk}, M, \sigma) = 1$, for all $\rho \in \mathsf{sample}_\rho$, for all $\tilde{\sigma} \in \mathsf{ConvertSig}(\mathsf{pk}, M, \sigma, \rho)$, $\mathsf{Verify}(\mathsf{ConvertPK}(\mathsf{pk}, \rho), M, \tilde{\sigma}) = 1$.

Change of message representative For all $M \in \mathcal{M}$, for all σ such that $\mathsf{Verify}(\mathsf{pk}, M, \sigma) = 1$, for all $\mu \in \mathsf{sample}_\mu$, $\mathsf{Verify}(\mathsf{pk}, M', \sigma') = 1$, where $(M', \sigma') = \mathsf{ChangeRep}(\mathsf{pk}, M, \sigma, \mu)$. Moreover, $M' \in [M]_{\mathcal{R}_M}$.

Let us discuss the intuition for the correctness property. Correct verification is simply the standard correctness property for signature schemes. Correct key conversion means that if the same key converter ρ is applied to a valid key pair $(\mathsf{pk}, \mathsf{sk})$, the result is a new valid key pair $(\tilde{\mathsf{pk}}, \tilde{\mathsf{sk}})$ from the same pair of equivalence classes. Correct signature conversion means that if the same key converter ρ is applied to a public key pk to obtain $\tilde{\mathsf{pk}}$ and to a valid signature σ on a message M to obtain $\tilde{\sigma}$, then the new signature $\tilde{\sigma}$ is a valid signature on the same message M under the new public key $\tilde{\mathsf{pk}}$. Finally, correct change of message representative ensures that if a message converter μ is applied to a valid message-signature pair (M, σ), the result is a new valid message-signature pair (M', σ'), where the new message M' is in the same equivalence class as M.

Definition 3 (Unforgeability) *A mercurial signature scheme* (PPGen, KeyGen, Sign, Verify, $\mathsf{ConvertSK}$, $\mathsf{ConvertPK}$, $\mathsf{ConvertSig}$, $\mathsf{ChangeRep}$) *for parameterized equivalence relations $\mathcal{R}_M$, $\mathcal{R}_{\mathsf{pk}}$, $\mathcal{R}_{\mathsf{sk}}$ is unforgeable if for all polynomial-length parameters $\ell(k)$ and all probabilistic, polynomial-time (PPT) algorithms $\mathcal{A}$ having access to a signing oracle, there exists a negligible function ν such that:*

$$\begin{aligned}\Pr[PP \leftarrow \mathsf{PPGen}(1^k); (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(PP, \ell(k)); (Q, \mathsf{pk}^*, M^*, \sigma^*) \leftarrow \\ \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot)}(\mathsf{pk}) : \forall M \in Q,\ [M^*]_{\mathcal{R}_M} \neq [M]_{\mathcal{R}_M} \wedge [\mathsf{pk}^*]_{\mathcal{R}_{\mathsf{pk}}} = [\mathsf{pk}]_{\mathcal{R}_{\mathsf{pk}}} \\ \wedge \mathsf{Verify}(\mathsf{pk}^*, M^*, \sigma^*) = 1] \leq \nu(k)\end{aligned}$$

where Q is the set of queries that $\mathcal{A}$ has issued to the signing oracle.

The unforgeability property here is similar to existential unforgeability (EUF-CMA) for signature schemes, except the adversary's winning condition is somewhat altered. As in the EUF-CMA game, the adversary is given the public key pk and is allowed to issue signature queries to the oracle that knows the corresponding secret key sk. Eventually, the adversary outputs a public key pk^*, a message M^*, and a purported signature σ^*. Unlike the EUF-CMA game, the adversary has the freedom to choose to output a forgery under a different public key pk^*, as long as pk^* is in the same equivalence class as pk. This seemingly makes the adversary's task easier. At the same time, the adversary's forgery is not valid if the message M^* is in the same equivalence class as a previously queried message, making the adversary's task harder. We can more formally relate our definitions to the standard definitions of existential unforgeability and correctness for signature schemes as follows. Suppose the relations $\mathcal{R}_M$, $\mathcal{R}_{\mathsf{pk}}$, $\mathcal{R}_{\mathsf{sk}}$ are equality relations (recall that $\mathcal{R}$ is an equality relation if $(a, b) \in \mathcal{R} \Leftrightarrow a = b$.) Let $\mathsf{ConvertSK}$, $\mathsf{ConvertPK}$, $\mathsf{ConvertSig}$, $\mathsf{ChangeRep}$ be algorithms that do nothing but simply output their input $\mathsf{sk}, \mathsf{pk}, \sigma, (M, \sigma)$, respectively. Then, it is easy to see that (PPGen, KeyGen, Sign, Verify) is a correct and existentially unforgeable signature scheme if and only if the mercurial signature scheme (PPGen, KeyGen,

Sign, Verify, ConvertSK, ConvertPK, ConvertSig, ChangeRep) for $\mathcal{R}_M$, $\mathcal{R}_{\mathsf{pk}}$, $\mathcal{R}_{\mathsf{sk}}$ is correct and unforgeable.

If one disregards insignificant differences in input-output specification and the emphasis on structure-preserving properties (not important for the security definition), our mercurial signatures are a generalization of Fuchsbauer, Hanser and Slamanig's [FHS14] structure-preserving signatures on equivalence classes (SPS-EQ) and in fact were inspired by signatures on equivalence classes in that paper. In an SPS-EQ signature for an equivalence relation $\mathcal{R}_M$, the ChangeRep algorithm is present, but there are no ConvertSK, ConvertPK, ConvertSig algorithms. The correctness requirement boils down to our correct verification and correct change of message representative requirements. Unforgeability of SPS-EQ is similar to unforgeability of mercurial signatures, except that $\mathcal{A}$ does not have the freedom to pick a different public key pk^*; the forgery must verify under the original public key pk. We can more formally relate our definitions to the definitions of existential unforgeability and correctness for signatures on equivalence classes as follows. Suppose the relations $\mathcal{R}_{\mathsf{pk}}$ and $\mathcal{R}_{\mathsf{sk}}$ are equality relations. Let ConvertSK, ConvertPK, ConvertSig be algorithms that do nothing but simply output their input $\mathsf{sk}, \mathsf{pk}, \sigma$, respectively. Then, (PPGen, KeyGen, Sign, Verify, ChangeRep) is a correct and unforgeable signature scheme for the equivalence relation $\mathcal{R}_M$ if and only if the mercurial signature scheme (PPGen, KeyGen, Sign, Verify, ConvertSK, ConvertPK, ConvertSig, ChangeRep) for $\mathcal{R}_M$, $\mathcal{R}_{\mathsf{pk}}$, $\mathcal{R}_{\mathsf{sk}}$ is correct and unforgeable.

Class- and origin-hiding of mercurial signatures. It is important for our application that the relations $\mathcal{R}_M$ and $\mathcal{R}_{\mathsf{pk}}$ be *class-hiding*. Class-hiding for messages [FHS14] means that given two messages, M_1 and M_2, it should be hard to tell whether or not $M_1 \in [M_2]_{\mathcal{R}_M}$. Class-hiding for public keys means that, given two public keys, pk_1 and pk_2, and oracle access to the signing algorithm for both of them, it is hard to tell whether or not $\mathsf{pk}_1 \in [\mathsf{pk}_2]_{\mathcal{R}_{\mathsf{pk}}}$.

An additional property we will need for our application is that, even if pk^* is adversarial, a message-signature pair obtained by running $\mathsf{ChangeRep}(\mathsf{pk}^*, M_0, \sigma_0, \mu_0)$ is distributed the same way as a pair obtained by running $\mathsf{ChangeRep}(\mathsf{pk}^*, M_1, \sigma_1, \mu_1)$, as long as M_0 and M_1 are in the same equivalence class. Thus, seeing the resulting message-signature pair hides its origin, whether it came from M_0 or M_1. Similarly, even for an adversarial pk^*, a signature on a message M output by ConvertSig hides whether ConvertSig was given pk^* as input or another pk in the same equivalence class.

Definition 4 *A mercurial signature scheme* (PPGen, KeyGen, Sign, Verify, ConvertSK, ConvertPK, ConvertSig, ChangeRep) *for parameterized equivalence relations $\mathcal{R}_M$, $\mathcal{R}_{\mathsf{pk}}$, $\mathcal{R}_{\mathsf{sk}}$ is class-hiding if it satisfies the following two properties:*

Message class-hiding: For all polynomial-length parameters $\ell(k)$ and all probabilistic polynomial-time (PPT) adversaries $\mathcal{A}$, there exists a negligible ν such that:

$$\Pr[PP \leftarrow \mathsf{PPGen}(1^k); M_1 \leftarrow \mathcal{M}; M_2^0 \leftarrow \mathcal{M}; M_2^1 \leftarrow [M_1]_{\mathcal{R}_M};$$

$$b \leftarrow \{0,1\}; b' \leftarrow \mathcal{A}(PP, M_1, M_2^b) \ : \ b' = b] \leq \frac{1}{2} + \nu(k)$$

Public key class-hiding: For all polynomial-length parameters $\ell(k)$ and all PPT adversaries $\mathcal{A}$, there exists a negligible ν such that:

$$\begin{aligned} \Pr[PP \leftarrow \mathsf{PPGen}(1^k); (\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{KeyGen}(PP, \ell(k)); \\ (\mathsf{pk}_2^0, \mathsf{sk}_2^0) \leftarrow \mathsf{KeyGen}(PP, \ell(k)); \\ \rho \leftarrow \mathsf{sample}_\rho(PP); \mathsf{pk}_2^1 = \mathsf{ConvertPK}(\mathsf{pk}_1, \rho); \mathsf{sk}_2^1 = \mathsf{ConvertSK}(\mathsf{sk}_1, \rho); \\ b \leftarrow \{0,1\}; b' \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}_1, \cdot), \mathsf{Sign}(\mathsf{sk}_2^b, \cdot)}(\mathsf{pk}_1, \mathsf{pk}_2^b) \ : \ b' = b] \leq \frac{1}{2} + \nu(k) \end{aligned}$$

A mercurial signature is also *origin-hiding* if the following two properties hold:

Origin-hiding of $\mathsf{ChangeRep}$**:** For all k, for all $PP \in \mathsf{PPGen}(1^k)$, for all pk^* (in particular, adversarially generated ones), for all M, σ, if $\mathsf{Verify}(\mathsf{pk}^*, M, \sigma) = 1$, if $\mu \leftarrow \mathsf{sample}_\mu$, then $\mathsf{ChangeRep}(\mathsf{pk}^*, M, \sigma, \mu)$ outputs a uniformly random $M' \in [M]_{\mathcal{R}_M}$ and a uniformly random $\sigma' \in \{\hat{\sigma} \mid \mathsf{Verify}(\mathsf{pk}^*, M', \hat{\sigma}) = 1\}$.

Origin-hiding of $\mathsf{ConvertSig}$**:** For all k, for all $PP \in \mathsf{PPGen}(1^k)$, for all pk^* (in particular, adversarially generated ones), for all M, σ, if $\mathsf{Verify}(\mathsf{pk}^*, M, \sigma) = 1$, if $\rho \leftarrow \mathsf{sample}_\rho$, then $\mathsf{ConvertSig}(\mathsf{pk}^*, M, \sigma, \rho)$ outputs a uniformly random $\tilde{\sigma} \in \{\hat{\sigma} \mid \mathsf{Verify}(\mathsf{ConvertPK}(\mathsf{pk}^*, \rho), M, \hat{\sigma}) = 1\}$, and $\mathsf{ConvertPK}(\mathsf{pk}^*, \rho)$ outputs a uniformly random element of $[\mathsf{pk}^*]_{\mathcal{R}_{\mathsf{pk}}}$ if $\rho \leftarrow \mathsf{sample}_\rho$.

3 Construction of Mercurial Signatures

Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a Type III bilinear pairing for multiplicative groups $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ of prime order p. Let $P, \hat{P}$, and $e(P, \hat{P})$ be generators, respectively (see Appendix A.1 for a review.) The message space for our mercurial signature scheme will consist of vectors of group elements from $\mathbb{G}_1^*$, where $\mathbb{G}_1^* = \mathbb{G}_1 \backslash \{1_{\mathbb{G}_1}\}$. The space of secret keys will consist of vectors of elements from $\mathbb{Z}_p^*$. The space of public keys, similar to the message space, will consist of vectors of group elements from $\mathbb{G}_2^*$. Once the prime p, $\mathbb{G}_1^*$, $\mathbb{G}_2^*$, and ℓ are well-defined, the equivalence relations of interest to us are as follows:

$$\begin{aligned} \mathcal{R}_M &= \{(M, M') \in (\mathbb{G}_1^*)^\ell \times (\mathbb{G}_1^*)^\ell \ \mid \ \exists r \in \mathbb{Z}_p^* \text{ such that } M' = M^r\} \\ \mathcal{R}_{\mathsf{sk}} &= \{(\mathsf{sk}, \tilde{\mathsf{sk}}) \in (\mathbb{Z}_p^*)^\ell \times (\mathbb{Z}_p^*)^\ell \ \mid \ \exists r \in \mathbb{Z}_p^* \text{ such that } \tilde{\mathsf{sk}} = \mathsf{sk}^r\} \\ \mathcal{R}_{\mathsf{pk}} &= \{(\mathsf{pk}, \tilde{\mathsf{pk}}) \in (\mathbb{G}_2^*)^\ell \times (\mathbb{G}_2^*)^\ell \ \mid \ \exists r \in \mathbb{Z}_p^* \text{ such that } \tilde{\mathsf{pk}} = \mathsf{pk}^r\} \end{aligned}$$

Note that messages, secret keys, and public keys are restricted to vectors consisting of only non-identity group elements. Without this restriction and the restriction that $r \neq 0$, the resulting relation would not be an equivalence one.

We introduce our mercurial signature construction with message space $(\mathbb{G}_1^*)^\ell$, but a mercurial signature scheme with message space $(\mathbb{G}_2^*)^\ell$ can be obtained by simply switching $\mathbb{G}_1^*$ and $\mathbb{G}_2^*$ throughout.

$\mathsf{PPGen}(1^k) \to PP$: Compute $\mathsf{BG} \leftarrow \mathsf{BGGen}(1^k)$. Output $PP = \mathsf{BG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, \hat{P}, e)$. Now that BG is well-defined, the relations $\mathcal{R}_M$, $\mathcal{R}_{\mathsf{pk}}$, $\mathcal{R}_{\mathsf{sk}}$ are also well-defined. sample_ρ and sample_μ are the same algorithm, namely the one that samples a random element of $\mathbb{Z}_p^*$.

$\mathsf{KeyGen}(PP, \ell) \to (\mathsf{pk}, \mathsf{sk})$: For $1 \leq i \leq \ell$, pick $x_i \leftarrow \mathbb{Z}_p^*$ and set secret key $\mathsf{sk} = (x_1, \ldots, x_\ell)$. Compute public key $\mathsf{pk} = (\hat{X}_1, \ldots, \hat{X}_\ell)$, where $\hat{X}_i = \hat{P}^{x_i}$ for $1 \leq i \leq \ell$. Output $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Sign}(\mathsf{sk}, M) \to \sigma$: On input $\mathsf{sk} = (x_1, \ldots, x_\ell)$ and $M = (M_1, \ldots, M_\ell) \in (\mathbb{G}_1^*)^\ell$, pick a random $y \leftarrow \mathbb{Z}_p^*$ and output $\sigma = (Z, Y, \hat{Y})$, where $Z \leftarrow \left(\prod_{i=1}^{\ell} M_i^{x_i}\right)^y$, $Y \leftarrow P^{\frac{1}{y}}$, and $\hat{Y} \leftarrow \hat{P}^{\frac{1}{y}}$.

$\mathsf{Verify}(\mathsf{pk}, M, \sigma) \to 0/1$: On input $\mathsf{pk} = (\hat{X}_1, \ldots, \hat{X}_\ell)$, $M = (M_1, \ldots, M_\ell)$, and $\sigma = (Z, Y, \hat{Y})$, check whether $\prod_{i=1}^{\ell} e(M_i, \hat{X}_i) = e(Z, \hat{Y}) \wedge e(Y, \hat{P}) = e(P, \hat{Y})$. If it holds, output 1; otherwise, output 0.

$\mathsf{ConvertSK}(\mathsf{sk}, \rho) \to \tilde{\mathsf{sk}}$: On input $\mathsf{sk} = (x_1, \ldots, x_\ell)$ and a key converter $\rho \in \mathbb{Z}_p^*$, output the new secret key $\tilde{\mathsf{sk}} = \mathsf{sk}^\rho$.

$\mathsf{ConvertPK}(\mathsf{pk}, \rho) \to \tilde{\mathsf{pk}}$: On input $\mathsf{pk} = (\hat{X}_1, \ldots, \hat{X}_\ell)$ and a key converter $\rho \in \mathbb{Z}_p^*$, output the new public key $\tilde{\mathsf{pk}} = \mathsf{pk}^\rho$.

$\mathsf{ConvertSig}(\mathsf{pk}, M, \sigma, \rho) \to \tilde{\sigma}$: On input pk, message M, signature $\sigma = (Z, Y, \hat{Y})$, and key converter $\rho \in \mathbb{Z}_p^*$, sample $\psi \leftarrow \mathbb{Z}_p^*$. Output $\tilde{\sigma} = (Z^{\psi\rho}, Y^{\frac{1}{\psi}}, \hat{Y}^{\frac{1}{\psi}})$.

$\mathsf{ChangeRep}(\mathsf{pk}, M, \sigma, \mu) \to (M', \sigma')$: On input pk, M, $\sigma = (Z, Y, \hat{Y})$, $\mu \in \mathbb{Z}_p^*$, sample $\psi \leftarrow \mathbb{Z}_p^*$. Compute $M' = M^\mu$, $\sigma' = (Z^{\psi\mu}, Y^{\frac{1}{\psi}}, \hat{Y}^{\frac{1}{\psi}})$. Output (M', σ').

Proofs of the following theorems can be found in Appendix B.

Theorem 1 (Correctness) *The construction described above is correct.*

Theorem 2 (Unforgeability) *The construction described above is unforgeable in the generic group model for Type III bilinear groups.*

To prove unforgeability, we construct a reduction to the unforgeability of the SPS-EQ signature scheme. Suppose a PPT algorithm $\mathcal{A}$ produces a successful forgery $(M^*, \sigma^*, \mathsf{pk}^*)$ for a mercurial signature scheme with non-negligible probability. Then, by definition, there exists some α in $\mathbb{Z}_p^*$ such that $\mathsf{pk}^* = \alpha\mathsf{pk}$, where pk is the challenge public key for unforgeability of SPS-EQ. We show that a PPT reduction $\mathcal{B}$ is able to obtain this α and produce a successful forgery $(\alpha M^*, \sigma^*, \mathsf{pk})$ for the SPS-EQ scheme, contradicting its proven security in the generic group model. The full proof can be found in Appendix B.2.

Theorem 3 (Class-hiding) *The construction described above is class-hiding in the generic group model for Type III bilinear groups.*

Message class-hiding follows from message class-hiding of the SPS-EQ scheme. Specifically, $(\mathbb{G}_i^*)^\ell$ is a class-hiding message space if and only if the decisional Diffie-Hellman assumption (DDH) holds in $\mathbb{G}_i$ [FHS14].

For public key class-hiding, we must show that an adversary's view in a game in which the challenger computes independent public keys $\mathsf{pk}_1 = (x_i^{(1)}\hat{X})_{i\in[\ell]}$ and $\mathsf{pk}_2 = (x_i^{(2)}\hat{X})_{i\in[\ell]}$ (Game 0) is the same as his view in a game in which $\mathsf{pk}_2 = \alpha\mathsf{pk}_1 = (\alpha x_i^{(1)}\hat{X})_{i\in[\ell]}$ for some $\alpha \in \mathbb{Z}_p^*$ (Game 3). We achieve this by constructing two intermediate games (Game 1 and Game 2). In Game 1, pk_1 and pk_2 are independent, but $\mathcal{C}$'s responses to $\mathcal{A}$'s group oracle and signing queries are computed as formal multivariate Laurent polynomials in the variables $x_1^{(1)}, ..., x_\ell^{(1)}, x_1^{(2)}, ..., x_\ell^{(2)}, y_1, ...y_q$, where y_i is the secret value the challenger uses for the i^{th} Sign query. In Game 2, $\mathsf{pk}_2 = \alpha\mathsf{pk}_1$ for some $\alpha \in \mathbb{Z}_p^*$, and $\mathcal{C}$'s responses to the oracle queries are again computed as formal multivariate Laurent polynomials, but now in the variables $x_1^{(1)}, ..., x_\ell^{(1)}, y_1, ...y_q$, and α. Demonstrating that $\mathcal{A}$'s view is the same in Game 0 as it is in Game 1 is a direct application of the Schwartz-Zippel lemma, which guarantees that the probability that a formal polynomial in Game 1, in which the variables $x_1^{(1)}, ..., x_\ell^{(1)}, x_1^{(2)}, ..., x_\ell^{(2)}, y_1, ...y_q$ are given to $\mathcal{A}$ as handles, collides with a formal polynomial in Game 0, in which the handles correspond to the variables that were fixed at the beginning of the game, is negligible. The same argument applies to Game 2 vs. Game 3.

It is nontrivial to prove that $\mathcal{A}$'s view is the same in Game 1 as it is in Game 2. First, we must show that for computations carried out by the challenger in each of the three groups, $\mathbb{G}_1^*, \mathbb{G}_2^*$, and $\mathbb{G}_T$, $\mathcal{A}$'s view is the same in both games. In $\mathbb{G}_2^*$, for example, we prove that two group oracle queries to $\mathbb{G}_2^*$ in Game 1 result in distinct formal polynomials if and only if the same two queries in Game 2 result in distinct polynomials. Then, the same must be shown, by induction, for signature queries too. If this sounds vague, it is because of the difficulty in conveying the details, which involve many variables and groups, in a high-level proof sketch. Please see Appendix B.3 for the full proof.

Theorem 4 (Origin-hiding) *The construction described above is origin-hiding in the generic group model for Type III bilinear groups.*

4 Definition of Delegatable Anonymous Credentials

Delegatable anonymous credentials have been studied before and previous definitions exist. The first paper to study the subject, due to Chase and Lysyanskaya [CL06], does not contain a definition of security. The next paper, by Belenkiy et al. [BCC+09], contains a simulation-extraction style definition. Anonym ity means there is a simulator that, when interacting with the adversary on behalf of honest parties, creates for each interaction a transcript whose distribution is independent on the identity of the honest party interacting with the adversary. The extractability part means there is an extractor that "de-anonymizes" the parties under the adversary's control and guarantees that the adversary cannot prove possession of a credential that "de-anonymizes" to a credential chain not corresponding to a sequence of credential issue instances that have actually occurred. A subsequent paper, by Chase et al. [CKLM13], suggested modifying the Belenkiy et al. definition but preserved the simulation-extraction style.

Our definitional approach is more traditional: we have a single security game, in which the adversary interacts with the system and attempts to break it either by forging a credential or de-anonymizing a user, or both. Thus, we do not rely on the definitional machinery of simulation and extraction that Belenkiy et al. "inherited" from non-interactive zero-knowledge proof of knowledge (NIZK PoK) systems. This makes our definition weaker than the Belenkiy at al. definition (as we will discuss below), but at the same time, doing away with simulation and extraction requirements means that it can be satisfied with cryptographic building blocks, such as mercurial signatures and (interactive) zero-knowledge proofs, that do not necessarily imply NIZK PoK.

Definition 5 (Delegatable anonymous credentials) *A delegatable anonymous credential scheme consists of algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{NymGen})$ *and protocols for issuing/receiving a credential and proving/verifying possession of a credential as follows:*

$\mathsf{Setup}(1^k) \to (params)$: A PPT algorithm that generates the public parameters *params* for the system.

$\mathsf{KeyGen}(params) \to (\mathsf{pk}, \mathsf{sk})$: A PPT algorithm that generates an "identity" of a system participant, which consists of a public and secret key pair $(\mathsf{pk}, \mathsf{sk})$. sk is referred to as the user's *secret identity key*, while pk is its *public identity key*. WLOG, sk is assumed to include both *params* and pk so that they need not be given to other algorithms as separate inputs. A root authority runs the same key generation algorithm as every other participant.

$\mathsf{NymGen}(\mathsf{sk}, L(\breve{\mathsf{pk}}_0)) \to (\mathsf{nym}, \mathsf{aux})$: A PPT algorithm that, on input a user's secret identity key sk and level $L(\breve{\mathsf{pk}}_0)$ under the root authority whose public key is $\breve{\mathsf{pk}}_0$, outputs a pseudonym nym for this user and the auxiliary information aux needed to use nym.

Issuing a credential:

$[\mathsf{Issue}(L_I(\breve{\mathsf{pk}}_0), \breve{\mathsf{pk}}_0, \mathsf{sk}_I, \mathsf{nym}_I, \mathsf{aux}_I, \mathsf{cred}_I, \mathsf{nym}_R) \leftrightarrow \mathsf{Receive}(L_I(\breve{\mathsf{pk}}_0), \breve{\mathsf{pk}}_0, \mathsf{sk}_R, \mathsf{nym}_R, \mathsf{aux}_R, \mathsf{nym}_I)] \to (\mathsf{cred}_R)$: This is an interactive protocol between an issuer of a credential, who runs the Issue side of the protocol, and a receiver, who runs the $\mathsf{Receive}$ side. The issuer takes as input his own credential at level $L_I(\breve{\mathsf{pk}}_0)$ under root authority $\breve{\mathsf{pk}}_0$ together with all information associated with it. Specifically, this includes $L_I(\breve{\mathsf{pk}}_0)$, the length of the issuer's credential chain; $\breve{\mathsf{pk}}_0$, the public key of the root authority; sk_I, the issuer's secret key; nym_I, the pseudonym by which the issuer is known to the receiver and its associated auxiliary information, aux_I; and cred_I, the issuer's credential chain. The issuer also takes as input nym_R, the pseudonym by which the receiver is known to him. The receiver takes as input the same $L_I(\breve{\mathsf{pk}}_0)$ and $\breve{\mathsf{pk}}_0$, the same nym_I and nym_R, her own secret key sk_R, and the auxiliary information aux_R associated with her pseudonym nym_R. The receiver's output is her credential cred_R.

Remarks. Note that there is a single protocol any issuer, including a root authority, runs with any recipient. A root authority does not use a pseudonym,

so our convention in that case is $L_I(\breve{\mathsf{pk}}_0) = 0$, $\mathsf{nym}_I = \breve{\mathsf{pk}}_0$, and $\mathsf{aux}_I = \mathsf{cred}_I = \perp$. Also, note that credentials, like levels, are dependent on $\breve{\mathsf{pk}}_0$ (i.e. $\mathsf{cred}_I = \mathsf{cred}_I(\breve{\mathsf{pk}}_0)$), but this dependency has been omitted for clarity.

Proof of possession of a credential:
$[\mathsf{CredProve}(L_P(\breve{\mathsf{pk}}_0), \breve{\mathsf{pk}}_0, \mathsf{sk}_P, \mathsf{nym}_P, \mathsf{aux}_P, \mathsf{cred}_P) \leftrightarrow \mathsf{CredVerify}(\mathit{params}, L_P(\breve{\mathsf{pk}}_0), \breve{\mathsf{pk}}_0, \mathsf{nym}_P)] \rightarrow$ output (0 or 1): This is an interactive protocol between a prover, who is trying to prove possession of a credential and runs the CredProve side of the protocol, and a verifier, who runs the CredVerify side. The prover takes as input his own credential at level $L_P(\breve{\mathsf{pk}}_0)$ under root authority $\breve{\mathsf{pk}}_0$ together with all information associated with it. Specifically, this includes $L_P(\breve{\mathsf{pk}}_0)$, the length of the prover's credential chain; $\breve{\mathsf{pk}}_0$, the public key of the root authority; sk_P, the prover's secret key; nym_P, the pseudonym by which the prover is known to the verifier and its associated auxiliary information, aux_P; and cred_P, the prover's credential chain. The verifier takes as input *params* and the same $L_P(\breve{\mathsf{pk}}_0)$, $\breve{\mathsf{pk}}_0$, and nym_P. The verifier's output is 1 if it accepts the proof of possession of a credential and 0 otherwise.

A delegatable anonymous credential (DAC) system must be correct and secure.

Definition 6 (Correctness of DAC) *A delegatable anonymous credential scheme is correct if, whenever* Setup, KeyGen *and* NymGen *are run correctly and the* Issue-Receive *protocol is executed correctly on correctly generated inputs, the receiver outputs a certification chain that, when used as input to the prover in an honest execution of the* CredProve-CredVerify *protocol, is accepted by the verifier with probability 1.*

We now provide a description of the security game along with a definition of unforgeability and anonymity for delegatable anonymous credentials under a single certification authority.

Security game. The security game is parameterized by (hard-to-compute) functions f, f_{cred}, and f_{demo}. An adversary $\mathcal{A}$ interacts with a challenger $\mathcal{C}$, who is responsible for setting up the keys and pseudonyms of all the honest participants in the system and for acting on their behalf when they issue, receive, prove possession of, or verify possession of credential chains. Throughout the game, $\mathcal{C}$ maintains the following state information:

1. A directed graph $G(\breve{\mathsf{pk}}_0) = (V(\breve{\mathsf{pk}}_0), E(\breve{\mathsf{pk}}_0))$ that will consist of a single tree and some singleton nodes. The root of the tree is the node called *root*, and it has public key $\breve{\mathsf{pk}}_0$.
2. Corresponding to every node $v \in V(\breve{\mathsf{pk}}_0)$, the following information:
   (a) v's level $L(\breve{\mathsf{pk}}_0, v)$ (i.e. v's distance to *root* $\breve{\mathsf{pk}}_0$).
   (b) $\mathit{status}(v)$, which specifies whether v corresponds to an honest or adversarial user.
   (c) If $\mathit{status}(v) = \mathit{honest}$, then

- $\mathsf{pk}(v)$, the public key associated with v;
- $\mathsf{sk}(v)$, the secret key corresponding to $\mathsf{pk}(v)$;
- all pseudonyms $\mathsf{nym}_1(v), \ldots, \mathsf{nym}_n(v)$ associated with v (if they exist) and their corresponding auxiliary information $\mathsf{aux}_1(v), \ldots, \mathsf{aux}_n(v)$;
- the user's credential $\mathsf{cred}_v := \mathsf{cred}_v(\breve{\mathsf{pk}}_0)$ (if it exists);
- a value $\hat{\mathsf{pk}}_v$, determined using the function f (as we will see, $\hat{\mathsf{pk}}_v = f(\mathsf{pk}(v)) = f(\mathsf{nym}_i(v))$ for $\mathsf{nym}_i(v) \in \{\mathsf{nym}_1(v), \ldots, \mathsf{nym}_n(v)\}$.)

(d) If $\mathit{status}(v) = \mathit{adversarial}$, a value $\hat{\mathsf{pk}}_v$, determined using the function f, that will be used as this node's identity. As we will see, $\hat{\mathsf{pk}}_v = f(\mathsf{nym}(v))$ if $\mathsf{nym}(v)$ is a pseudonym used by the adversary on behalf of node v. Note that for two different adversarial nodes v, v', it is possible that $\hat{\mathsf{pk}}_v = \hat{\mathsf{pk}}_{v'}$. This is not possible for honest nodes.

3. A `forgery` flag, which is set to true if the adversary forges a credential.
4. An anonymity bit $b \in \{0, 1\}$, a pair of anonymity challenge nodes (u_0, u_1), and the status of the anonymity attack. Define S to be the set of pairs of pseudonyms $(\mathsf{nym}(u_b), \mathsf{nym}(u_{\bar{b}}))$ that the adversary has seen for the anonymity challenge node u_b and the other node in the pair, $u_{\bar{b}}$, where $\bar{b} = 1 - b$. The challenger keeps track of S along with the auxiliary information for the pairs of pseudonyms it contains.

The security game is initialized as follows. The *params* are generated and given to the adversary $\mathcal{A}$. Then, $\mathcal{A}(\mathit{params})$ specifies whether the status of the root node is going to be *honest* or *adversarial*. If it is *honest*, the challenger $\mathcal{C}$ generates the root key pair, $(\breve{\mathsf{pk}}_0, \breve{\mathsf{sk}}_0) \leftarrow \mathsf{KeyGen}(\mathit{params})$; else, $\mathcal{A}$ supplies $\breve{\mathsf{pk}}_0$ to $\mathcal{C}$. Next, $\mathcal{C}$ sets the `forgery` flag to false and picks a random value for the anonymity bit b: $b \leftarrow \{0, 1\}$. At this point, the anonymity attack has not begun yet, so its status is *undefined*. $\mathcal{C}$ stores $G(\breve{\mathsf{pk}}_0) = (V(\breve{\mathsf{pk}}_0), E(\breve{\mathsf{pk}}_0)) = (\{\mathit{root}\}, \emptyset)$ (i.e. the graph consisting of the root node and no edges) and sets $\mathit{status}(\mathit{root})$ to be as specified by $\mathcal{A}$: $\mathsf{pk}(\mathit{root}) = \breve{\mathsf{pk}}_0$, and when $\mathit{status}(\mathit{root}) = \mathit{honest}$, $\mathsf{sk}(\mathit{root}) = \breve{\mathsf{sk}}_0$. Next, $\mathcal{A}$ can add nodes/users to $G(\breve{\mathsf{pk}}_0)$, both honest and adversarial, and have these users obtain, delegate, and prove possession of credentials by interacting with $\mathcal{C}$ using the following oracles:

$\mathsf{AddHonestParty}(u)$: $\mathcal{A}$ invokes this oracle to create a new, honest node u. $\mathcal{C}$ runs $(\mathsf{pk}(u), \mathsf{sk}(u)) \leftarrow \mathsf{KeyGen}(\mathit{params})$, sets $L(\breve{\mathsf{pk}}_0, u) = \infty$, and returns $\mathsf{pk}(u)$ to $\mathcal{A}$.

$\mathsf{SeeNym}(u)$: $\mathcal{A}$ invokes this oracle to see a fresh pseudonym for an honest node u. $\mathcal{C}$ runs $(\mathsf{nym}(u), \mathsf{aux}(u)) \leftarrow \mathsf{NymGen}(\mathsf{sk}(u), L(\breve{\mathsf{pk}}_0, u))$ and returns $\mathsf{nym}(u)$ to $\mathcal{A}$.

$\mathsf{CertifyHonestParty}(\breve{\mathsf{pk}}_0, u, v)$: $\mathcal{A}$ invokes this oracle to have the honest party associated with u issue a credential to the honest party associated with v. $\mathcal{A}$ selects pseudonyms $\mathsf{nym}(u), \mathsf{nym}(v)$ that he has seen for u, v (unless $u = \mathit{root}$), and $\mathcal{C}$ runs the protocols: $[\mathsf{Issue}(L(\breve{\mathsf{pk}}_0, u), \breve{\mathsf{pk}}_0, \mathsf{sk}(u), \mathsf{nym}(u), \mathsf{aux}(u), \mathsf{cred}_u, \mathsf{nym}(v)) \leftrightarrow \mathsf{Receive}(L(\breve{\mathsf{pk}}_0, u), \breve{\mathsf{pk}}_0, \mathsf{sk}(v), \mathsf{nym}(v), \mathsf{aux}(v), \mathsf{nym}(u))] \rightarrow \mathsf{cred}_v$. If $u = \mathit{root}$, then $\breve{\mathsf{pk}}_0$ is given as input instead of $\mathsf{nym}(u)$. $\mathcal{C}$ adds the edge (u, v) to the graph and sets $L(\breve{\mathsf{pk}}_0, v) = L(\breve{\mathsf{pk}}_0, u) + 1$.

$\mathsf{VerifyCredFrom}(\breve{\mathsf{pk}}_0, u)$: The honest party associated with u proves to $\mathcal{A}$ that it has a credential at level $L(\breve{\mathsf{pk}}_0, u)$. $\mathcal{A}$ selects a pseudonym $\mathsf{nym}(u)$ that he has seen for u, and $\mathcal{C}$ runs the $\mathsf{CredProve}$ protocol with $\mathcal{A}$: $\mathsf{CredProve}(L(\breve{\mathsf{pk}}_0, u), \breve{\mathsf{pk}}_0, \mathsf{sk}(u), \mathsf{nym}(u), \mathsf{aux}(u), \mathsf{cred}_u) \leftrightarrow \mathcal{A}$.

$\mathsf{GetCredFrom}(\breve{\mathsf{pk}}_0, u, \mathsf{nym}_R)$: The honest party associated with u issues a credential to $\mathcal{A}$, whom it knows by nym_R. $\mathcal{C}$ creates a new adversarial node v and sets its identity to be $\hat{\mathsf{pk}}_v = f(\mathsf{nym}_R)$. $\mathcal{A}$ selects a pseudonym $\mathsf{nym}(u)$ that he has seen for u (unless $u = root$), and $\mathcal{C}$ runs the Issue protocol with $\mathcal{A}$: $\mathsf{Issue}(L(\breve{\mathsf{pk}}_0, u), \breve{\mathsf{pk}}_0, \mathsf{sk}(u), \mathsf{nym}(u), \mathsf{aux}(u), \mathsf{cred}_u, \mathsf{nym}_R) \leftrightarrow \mathcal{A}$. If $u = root$, then $\breve{\mathsf{pk}}_0$ is given as input instead of $\mathsf{nym}(u)$. $\mathcal{C}$ adds the edge (u, v) to the graph and sets $L(\breve{\mathsf{pk}}_0, v) = L(\breve{\mathsf{pk}}_0, u) + 1$.

$\mathsf{GiveCredTo}(\breve{\mathsf{pk}}_0, L_I(\breve{\mathsf{pk}}_0), \mathsf{nym}_I, v)$: $\mathcal{A}$ issues a credential to the honest party associated with v under a pseudonym nym_I (or $\breve{\mathsf{pk}}_0$ if he is the root). $\mathcal{A}$ selects a pseudonym $\mathsf{nym}(v)$ that he has seen for v, and $\mathcal{C}$ runs the $\mathsf{Receive}$ protocol with $\mathcal{A}$: $[\mathcal{A} \leftrightarrow \mathsf{Receive}(L_I(\breve{\mathsf{pk}}_0), \breve{\mathsf{pk}}_0, \mathsf{sk}(v), \mathsf{nym}(v), \mathsf{aux}(v), \mathsf{nym}_I)] \rightarrow \mathsf{cred}_v$. If $\mathcal{A}$ is the *root*, then $\breve{\mathsf{pk}}_0$ is given as input instead of nym_I. If $\mathsf{cred}_v \neq \perp$, $\mathcal{C}$ sets $L(\breve{\mathsf{pk}}_0, v) = L_I(\breve{\mathsf{pk}}_0) + 1$ and computes the function f_{cred} on v's credential, $f_{cred}(\mathsf{cred}_v) = (\hat{\mathsf{pk}}_0, \hat{\mathsf{pk}}_1, \ldots, \hat{\mathsf{pk}}_{L_I})$, revealing the identities in v's credential chain. If according to $\mathcal{C}$'s data structure, there is some $\hat{\mathsf{pk}}_i$ in this chain such that $\hat{\mathsf{pk}}_i = f(\mathsf{nym}(u))$ for an honest user u, but $\hat{\mathsf{pk}}_{i+1} \neq f(\mathsf{nym}(v'))$ for any v' that received a credential from u, then $\mathcal{C}$ sets the `forgery` flag to true. If $\mathsf{cred}_v \neq \perp$ and the `forgery` flag remains false, $\mathcal{C}$ fills in the gaps in the graph as follows. Starting from the nearest honest ancestor of v, $\mathcal{C}$ creates a new node for each (necessarily adversarial) identity in the chain between that honest node and v and sets its identity to be the appropriate $\hat{\mathsf{pk}}_j$. $\mathcal{C}$ then adds edges between the nodes on the chain from the nearest honest ancestor of v to v.

$\mathsf{DemoCred}(\breve{\mathsf{pk}}_0, L_P(\breve{\mathsf{pk}}_0), \mathsf{nym}_P)$: $\mathcal{A}$ proves possession of a credential at level $L_P(\breve{\mathsf{pk}}_0)$. $\mathcal{C}$ runs the Verify protocol with $\mathcal{A}$: $[\mathcal{A} \leftrightarrow \mathsf{CredVerify}(params, L_P(\breve{\mathsf{pk}}_0), \breve{\mathsf{pk}}_0, \mathsf{nym}_P)] \rightarrow$ output (0 or 1). If output $= 1$, $\mathcal{C}$ computes the function f_{demo} on the transcript of the output, $f_{demo}(transcript) = (\hat{\mathsf{pk}}_0, \hat{\mathsf{pk}}_1, \ldots, \hat{\mathsf{pk}}_{L_P})$, and determines if a forgery has occurred as in $\mathsf{GiveCredTo}$. If output $= 1$ and the `forgery` flag remains false, $\mathcal{C}$ creates a new adversarial node v for the identity pk_{L_P} and sets $L(\breve{\mathsf{pk}}_0, v) = L_P(\breve{\mathsf{pk}}_0)$. $\mathcal{C}$ fills in the gaps in the graph as in $\mathsf{GiveCredTo}$.

$\mathsf{SetAnonChallenge}(u_0, u_1)$: $\mathcal{A}$ will try to distinguish between the honest parties associated with u_0 and u_1.

$\mathsf{SeeNymAnon}$: $\mathcal{A}$ invokes this oracle to see fresh pseudonyms for u_b and $u_{\bar{b}}$. $\mathcal{C}$ runs $(\mathsf{nym}(u_b), \mathsf{aux}(u_b)) \leftarrow \mathsf{NymGen}(\mathsf{sk}(u_b), L(\breve{\mathsf{pk}}_0, u_b))$, repeats this for $u_{\bar{b}}$, and returns $(\mathsf{nym}(u_b), \mathsf{nym}(u_{\bar{b}}))$ to $\mathcal{A}$.

$\mathsf{CertifyHonestAnon}(\breve{\mathsf{pk}}_0, u)$: $\mathcal{A}$ invokes this oracle to have the honest party associated with u issue credentials to u_b and $u_{\bar{b}}$. $\mathcal{A}$ selects pseudonyms $(\mathsf{nym}(u_b), \mathsf{nym}(u_{\bar{b}}))$ and $\mathsf{nym}(u)$ that he has seen for $u_b, u_{\bar{b}}$, and u, and $\mathcal{C}$ runs the protocols: $[\mathsf{Issue}(L(\breve{\mathsf{pk}}_0, u), \breve{\mathsf{pk}}_0, \mathsf{sk}(u), \mathsf{nym}(u), \mathsf{aux}(u), \mathsf{cred}_u, \mathsf{nym}(u_b)) \leftrightarrow \mathsf{Receive}($

$L(\breve{\mathsf{pk}}_0, u), \breve{\mathsf{pk}}_0, \mathsf{sk}(u_b), \mathsf{nym}(u_b), \mathsf{aux}(u_b), \mathsf{nym}(u))] \rightarrow \mathsf{cred}_{u_b}$. If $u = root$, then $\breve{\mathsf{pk}}_0$ is given as input instead of $\mathsf{nym}(u)$. $\mathcal{C}$ adds the edge (u, u_b) to the graph and sets $L(\breve{\mathsf{pk}}_0, u_b) = L(\breve{\mathsf{pk}}_0, u) + 1$. $\mathcal{C}$ repeats these steps for $u_{\bar{b}}$, using the same $\mathsf{nym}(u)$ (if $u \neq root$).

$\mathsf{CertifyAnonHonest}(\breve{\mathsf{pk}}_0, b^*, v)$: $\mathcal{A}$ invokes this oracle to have one of the anonymity challenge nodes, u_{b^*}, where $b^* = b$ or $\bar{b}$, issue a credential to the honest party associated with v. $\mathcal{A}$ selects pseudonyms $(\mathsf{nym}(u_b), \mathsf{nym}(u_{\bar{b}}))$ and $\mathsf{nym}(v)$ that he has seen for $u_b, u_{\bar{b}}$, and v, and $\mathcal{C}$ runs the protocols: $[\mathsf{Issue}(L(\breve{\mathsf{pk}}_0, u_{b^*}), \breve{\mathsf{pk}}_0, \mathsf{sk}(u_{b^*}), \mathsf{nym}(u_{b^*}), \mathsf{aux}(u_{b^*}), \mathsf{cred}_{u_{b^*}}, \mathsf{nym}(v)) \leftrightarrow \mathsf{Receive}(L(\breve{\mathsf{pk}}_0, u_{b^*}), \breve{\mathsf{pk}}_0, \mathsf{sk}(v), \mathsf{nym}(v), \mathsf{aux}(v), \mathsf{nym}(u_{b^*}))] \rightarrow \mathsf{cred}_v$. $\mathcal{C}$ adds the edge (u_{b^*}, v) to the graph and sets $L(\breve{\mathsf{pk}}_0, v) = L(\breve{\mathsf{pk}}_0, u_{b^*}) + 1$.

$\mathsf{VerifyCredFromAnon}(\breve{\mathsf{pk}}_0)$: The honest parties associated with u_b and $u_{\bar{b}}$ prove to $\mathcal{A}$ that they have credentials at level $L(\breve{\mathsf{pk}}_0, u_b) = L(\breve{\mathsf{pk}}_0, u_{\bar{b}})$. $\mathcal{C}$ checks that the two paths from u_b and $u_{\bar{b}}$ to the root $\breve{\mathsf{pk}}_0$ consist entirely of honest nodes, with the exception that $\breve{\mathsf{pk}}_0$ may be adversarial. If this check fails, $\mathcal{C}$ updates the status of the anonymity attack to *forfeited*. Next, $\mathcal{A}$ selects pseudonyms $(\mathsf{nym}(u_b), \mathsf{nym}(u_{\bar{b}}))$ that he has seen for u_b and $u_{\bar{b}}$, and $\mathcal{C}$ runs the $\mathsf{CredProve}$ protocol with $\mathcal{A}$: $\mathsf{CredProve}(L(\breve{\mathsf{pk}}_0, u_b), \breve{\mathsf{pk}}_0, \mathsf{sk}(u_b), \mathsf{nym}(u_b), \mathsf{aux}(u_b), \mathsf{cred}_{u_b}) \leftrightarrow \mathcal{A}$. $\mathcal{C}$ repeats this step for $u_{\bar{b}}$.

$\mathsf{GetCredFromAnon}(\breve{\mathsf{pk}}_0, b^*, \mathsf{nym}_R)$: The honest party associated with u_{b^*}, where $b^* = b$ or $\bar{b}$, issues a credential to $\mathcal{A}$, whom it knows by nym_R. $\mathcal{C}$ checks the two paths from u_{b^*} and $u_{\bar{b}^*}$ to the root $\breve{\mathsf{pk}}_0$ as in $\mathsf{VerifyCredFromAnon}$. Next, $\mathcal{C}$ creates a new adversarial node v and sets its identity to be $\hat{\mathsf{pk}}_v = f(\mathsf{nym}_R)$. Note that $\mathcal{A}$ can have $u_{b^*}, u_{\bar{b}^*}$ issue credentials to two different adversarial nodes v, v', respectively, with the same underlying adversarial identity $\hat{\mathsf{pk}}_v = \hat{\mathsf{pk}}_{v'}$. $\mathcal{A}$ selects pseudonyms $(\mathsf{nym}(u_{b^*}), \mathsf{nym}(u_{\bar{b}^*}))$ that he has seen for u_{b^*} and $u_{\bar{b}^*}$, and $\mathcal{C}$ runs the Issue protocol with $\mathcal{A}$: $\mathsf{Issue}(L(\breve{\mathsf{pk}}_0, u_{b^*}), \breve{\mathsf{pk}}_0, \mathsf{sk}(u_{b^*}), \mathsf{nym}(u_{b^*}), \mathsf{aux}(u_{b^*}), \mathsf{cred}_{u_{b^*}}, \mathsf{nym}_R) \leftrightarrow \mathcal{A}$. $\mathcal{C}$ adds the edge (u_{b^*}, v) to the graph and sets $L(\breve{\mathsf{pk}}_0, v) = L(\breve{\mathsf{pk}}_0, u_{b^*}) + 1$.

$\mathsf{GiveCredToAnon}(\breve{\mathsf{pk}}_0, L_I(\breve{\mathsf{pk}}_0), \mathsf{nym}_I)$: $\mathcal{A}$ issues credentials to u_b and $u_{\bar{b}}$ under a pseudonym nym_I (or $\breve{\mathsf{pk}}_0$ if he is the root). $\mathcal{A}$ selects pseudonyms $(\mathsf{nym}(u_b), \mathsf{nym}(u_{\bar{b}}))$ that he has seen for u_b and $u_{\bar{b}}$, and $\mathcal{C}$ runs the $\mathsf{Receive}$ protocol with $\mathcal{A}$: $[\mathcal{A} \leftrightarrow \mathsf{Receive}(L_I(\breve{\mathsf{pk}}_0), \breve{\mathsf{pk}}_0, \mathsf{sk}(u_b), \mathsf{nym}(u_b), \mathsf{aux}(u_b), \mathsf{nym}_I)] \rightarrow \mathsf{cred}_{u_b}$. If $\mathcal{A}$ is the root, then $\breve{\mathsf{pk}}_0$ is given as input instead of nym_I. $\mathcal{C}$ repeats this step for $u_{\bar{b}}$. If both $\mathsf{cred}_{u_b} \neq \perp$ and $\mathsf{cred}_{u_{\bar{b}}} \neq \perp$, $\mathcal{C}$ sets $L(\breve{\mathsf{pk}}_0, u_b) = L_I(\breve{\mathsf{pk}}_0) + 1$, computes the function f_{cred} on u_b's credential, $f_{cred}(\mathsf{cred}_{u_b}) = (\hat{\mathsf{pk}}_0, \hat{\mathsf{pk}}_1, \ldots, \hat{\mathsf{pk}}_{L_I})$, and determines if a forgery has occurred as in $\mathsf{GiveCredTo}$. $\mathcal{C}$ repeats this step for $u_{\bar{b}}$. If both $\mathsf{cred}_{u_b} \neq \perp$ and $\mathsf{cred}_{u_{\bar{b}}} \neq \perp$ and the `forgery` flag remains false, $\mathcal{C}$ fills in the gaps in the graph as follows. If there is already an adversarial node v corresponding to the pseudonym nym_I with an edge connecting it to an honest parent, then $\mathcal{C}$ only adds an edge between v and u_b. Else, $\mathcal{C}$ creates a chain of edges and (adversarial) nodes from the nearest honest ancestor of u_b to u_b as in $\mathsf{GiveCredTo}$. $\mathcal{C}$ repeats this step for $u_{\bar{b}}$.

$\mathsf{GuessAnon}(b')$: If $b' = b$, the status of the anonymity attack is set to *success*.

Definition 7 (Unforgeability and Anonymity) *A delegatable anonymous credential scheme is unforgeable and anonymous if there exist functions f, f_{cred}, and f_{demo} such that for all PPT $\mathcal{A}$, there exists a negligible function ν such that:*

1. *the probability that the* `forgery` *flag will be* true *in the single-authority game is at most $\nu(k)$, where k is the security parameter.*
2. *the probability that the status of the anonymity attack in the single-authority game will be success is at most $1/2 + \nu(k)$.*

Strengthening anonymity. Note that this flavor of anonymity is weaker than the previous one by Belenkiy et al., and not only because it is not based on simulatability. In our anonymity game, two nodes, u_0 and u_1, may have credentials at the same level but still may not be appropriate candidates for the anonymity challenge; the two paths from u_0 and u_1 to the root must consist entirely of honest nodes, with the exception that the root may be adversarial. The reason is that our definition allows the adversary to recognize himself on a credential chain, so if he were on u_0's credential chain but not u_1's, he would be able to distinguish the two. It would be relatively straightforward to "fix" our definition to not allow this: we would just need to remove the requirement that u_0 and u_1 have entirely honest paths to the root. Unfortunately, our construction only satisfies the weaker anonymity notion in which this requirement must be met.

Other types of composition. Note that in our definition, the adversary invokes the oracles *sequentially*. Our definition gives no security guarantees when the oracles are invoked concurrently and the adversary orchestrates the order in which protocol messages are exchanged. There are standard techniques for turning certain sequentially secure protocols into concurrently secure ones; there are also well-known subtleties [Dam00,Lin03b,Lin03a]. As far as stronger types of composition, such as universal composition [Can01], are concerned, our definition, even if modified somewhat, does not seem to provide this level of security.

5 Construction of DAC from Mercurial Signatures

Suppose we have a mercurial signature scheme such that the space of public keys is a subset of the message space and $\mathcal{R}_M = \mathcal{R}_{\mathsf{pk}}$. Furthermore, suppose it has the property that $\mathsf{sample}_\mu = \mathsf{sample}_\rho$ and on input $\mathsf{ChangeRep}(\mathsf{pk}, M, \sigma, \mu)$, where $M = \mathsf{pk}'$, it outputs (M', σ') such that $M' = \mathsf{ConvertPK}(\mathsf{pk}', \mu)$.

To generate a key pair, each participant runs the KeyGen algorithm of the mercurial signature scheme to get $(\mathsf{pk}, \mathsf{sk})$. To generate a new pseudonym and its auxiliary information, pick ρ and let $\mathsf{nym} = \mathsf{ConvertPK}(\mathsf{pk}, \rho)$, $\mathsf{aux} = \rho$.

A credential chain of length L will consist of a series of pseudonyms $(\mathsf{nym}_1, \ldots, \mathsf{nym}_L)$ and a series of signatures $(\sigma_1, \ldots, \sigma_L)$ such that (1) σ_1 is a signature on the message nym_1 under the certification authority's public key $\breve{\mathsf{pk}}_0$; (2) for $2 \leq i \leq L$, σ_i is the signature on the message nym_i under the public key nym_{i-1}. This is possible because the message space contains the space of public keys and the relations $\mathcal{R}_M$ and $\mathcal{R}_{\mathsf{pk}}$ are the same.

A credential chain can be randomized so as to be unrecognizable by using the $\mathsf{ConvertSig}$ and $\mathsf{ChangeRep}$ algorithms as follows. The input to this step is $(\mathsf{nym}_1, \ldots, \mathsf{nym}_L)$ and $(\sigma_1, \ldots, \sigma_L)$. In order to randomize it, pick random $(\rho_1, \ldots, \rho_L) \leftarrow \mathsf{sample}_\rho^L$. Define $\mathsf{nym}'_0 = \breve{\mathsf{pk}}_0$, $\tilde{\sigma}_1 = \sigma_1$. Now, perform two steps: (1) for $2 \leq i \leq L$, set $\tilde{\sigma}_i = \mathsf{ConvertSig}(\mathsf{nym}_{i-1}, \mathsf{nym}_i, \sigma_i, \rho_{i-1})$, and (2) for $1 \leq i \leq L$, set $(\mathsf{nym}'_i, \sigma'_i) = \mathsf{ChangeRep}(\mathsf{nym}'_{i-1}, \mathsf{nym}_i, \tilde{\sigma}_i, \rho_i)$. This way, nym'_i is the new, unrecognizable pseudonym corresponding to the same underlying identity as nym_i, and σ'_i is a signature attesting to that fact, which verifies under the already updated pseudonym nym'_{i-1} (treated as a public key for the purposes of message verification). Finally, output $(\mathsf{nym}'_1, \ldots, \mathsf{nym}'_L)$ and $(\sigma'_1, \ldots, \sigma'_L)$.

In order to issue a credential, the issuer first has the receiver prove, via an interactive zero-knowledge proof of knowledge (ZKPoK), that the receiver knows the secret key associated with his pseudonym, nym_R. Then, the issuer randomizes his certification chain as described above and uses the last pseudonym on the randomized chain, nym'_L, as his issuer's pseudonym, nym_I (alternatively, he can give a zero-knowledge proof that the two are equivalent.) He then computes $\sigma_{L+1} = \mathsf{Sign}(\mathsf{sk}_I, \mathsf{nym}_R)$, where sk_I is the secret key that corresponds to nym_I. He sends the randomized chain as well as σ_{L+1} to the receiver, who stores the resulting credential chain $(\mathsf{nym}'_1, \ldots, \mathsf{nym}'_L, \mathsf{nym}_R)$ and $(\sigma'_1, \ldots, \sigma'_L, \sigma_{L+1})$. In order to prove possession of a credential, a prover first randomizes the credential chain, reveals it to the verifier, and proves knowledge of the secret key that corresponds to the last pseudonym, nym'_L, on the certification chain.

Unfortunately, we do not know of a construction of a mercurial signature in which the public key space is a subset of the message space. Our mercurial signature construction does not enjoy that property because messages are vectors in $\mathbb{G}_1^*$, while public keys are vectors in $\mathbb{G}_2^*$. However, we know how to construct a pair of mercurial signature schemes in which the public key space of one is the message space of the other, and vice versa. That is accomplished by just switching $\mathbb{G}_1^*$ and $\mathbb{G}_2^*$ in one of the schemes; the secret key space is the same in both. We can use this pair of mercurial signature schemes to construct delegatable anonymous credentials similar to the intuitive way described above, except that we must invoke different algorithms for even positions on the certification chain than we do for odd positions.

Let $\mathsf{MS}_1 = (\mathsf{PPGen}_1, \mathsf{KeyGen}_1, \mathsf{Sign}_1, \mathsf{Verify}_1, \mathsf{ConvertSK}_1, \mathsf{ConvertPK}_1, \mathsf{ConvertSig}_1, \mathsf{ChangeRep}_1)$ and $\mathsf{MS}_2 = (\mathsf{PPGen}_2, \mathsf{KeyGen}_2, \mathsf{Sign}_2, \mathsf{Verify}_2, \mathsf{ConvertSK}_2, \mathsf{ConvertPK}_2, \mathsf{ConvertSig}_2, \mathsf{ChangeRep}_2)$ be two mercurial signature schemes that share the same parameter generation algorithm $\mathsf{PPGen}_1 = \mathsf{PPGen}_2$. Let $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_{\mathsf{sk}}$ be parameterized relations such that MS_1 has message relation $\mathcal{R}_1$, public key relation $\mathcal{R}_2$, and secret key relation $\mathcal{R}_{\mathsf{sk}}$, while MS_2 has message relation $\mathcal{R}_2$, public key relation $\mathcal{R}_1$, and the same secret key relation $\mathcal{R}_{\mathsf{sk}}$. Suppose $\mathsf{sample}_\mu = \mathsf{sample}_\rho$ for both schemes and that the message space for the first scheme consists of public keys for the second one, and vice versa. Finally, suppose that both schemes satisfy class- and origin-hiding.

Our construction consists of the following algorithms and protocols. Initially, a user runs KeyGen to obtain two key pairs, an odd pair and an even pair. Once

a user receives a credential, her level is fixed, so she only uses the relevant key pair - the odd pair to be used at an odd level and the even pair at an even level.

$\mathsf{Setup}(1^k) \to (params)$: Compute $PP \leftarrow \mathsf{PPGen}_1(1^k) = \mathsf{PPGen}_2(1^k)$ and output $params = PP$.

$\mathsf{KeyGen}(params) \to (\mathsf{pk}, \mathsf{sk})$: There are two cases. For the root authority, compute $(\breve{\mathsf{pk}}_0, \breve{\mathsf{sk}}_0) \leftarrow \mathsf{KeyGen}_1(PP, \ell)$ and output it. For others, compute $(\mathsf{pk}_{\mathsf{even}}, \mathsf{sk}_{\mathsf{even}}) \leftarrow \mathsf{KeyGen}_1(PP, \ell)$ and $(\mathsf{pk}_{\mathsf{odd}}, \mathsf{sk}_{\mathsf{odd}}) \leftarrow \mathsf{KeyGen}_2(PP, \ell)$ and output both pairs of keys $(\mathsf{pk}_{\mathsf{even}}, \mathsf{sk}_{\mathsf{even}}), (\mathsf{pk}_{\mathsf{odd}}, \mathsf{sk}_{\mathsf{odd}})$.

$\mathsf{NymGen}(\mathsf{sk}, L(\breve{\mathsf{pk}}_0)) \to (\mathsf{nym}, \mathsf{aux})$: If $L(\breve{\mathsf{pk}}_0) = 0$, output $(\breve{\mathsf{pk}}_0, \perp)$. Otherwise, pick random key converters $\rho_{\mathsf{even}}, \rho_{\mathsf{odd}}$ and compute $\tilde{\mathsf{sk}}_{\mathsf{even}} \leftarrow \mathsf{ConvertSK}_1(\mathsf{sk}_{\mathsf{even}}, \rho_{\mathsf{even}})$ and $\mathsf{nym}_{\mathsf{even}} \leftarrow \mathsf{ConvertPK}_1(\mathsf{pk}_{\mathsf{even}}, \rho_{\mathsf{even}})$. Similarly, compute $\tilde{\mathsf{sk}}_{\mathsf{odd}} \leftarrow \mathsf{ConvertSK}_2(\mathsf{sk}_{\mathsf{odd}}, \rho_{\mathsf{odd}})$ and $\mathsf{nym}_{\mathsf{odd}} \leftarrow \mathsf{ConvertPK}_2(\mathsf{pk}_{\mathsf{odd}}, \rho_{\mathsf{odd}})$. Output both pairs $(\mathsf{nym}_{\mathsf{even}}, \rho_{\mathsf{even}}), (\mathsf{nym}_{\mathsf{odd}}, \rho_{\mathsf{odd}})$.

In the following protocols, each algorithm is either from MS_1 or MS_2, but the even/odd subscripts have been omitted for clarity. For example, $\sigma_1 \leftarrow \mathsf{Sign}(\breve{\mathsf{sk}}_0, \mathsf{nym}_1)$ is computed as $\sigma_1 \leftarrow \mathsf{Sign}_1(\breve{\mathsf{sk}}_0, \mathsf{nym}_{1,\mathsf{odd}})$ since the user nym_1 is fixed at odd level 1.

Issuing a credential:

$\mathsf{Issue}(L_I(\breve{\mathsf{pk}}_0), \breve{\mathsf{pk}}_0, \mathsf{sk}_I, \mathsf{nym}_I, \mathsf{aux}_I, \mathsf{cred}_I, \mathsf{nym}_R) \leftrightarrow \mathsf{Receive}(L_I(\breve{\mathsf{pk}}_0), \breve{\mathsf{pk}}_0, \mathsf{sk}_R, \mathsf{nym}_R, \mathsf{aux}_R, \mathsf{nym}_I)] \to \mathsf{cred}_R$: $\{(\mathsf{nym}'_1, \ldots, \mathsf{nym}'_{L_I}, \mathsf{nym}_R), (\sigma'_1, \ldots, \sigma'_{L_I}, \sigma_{L_I+1})\}$
The issuer first has the receiver prove, via an interactive ZKPoK, that the receiver knows the secret key, sk_R, associated with his pseudonym, nym_R.

Issue: *Randomize credential chain.*

1. If $L_I(\breve{\mathsf{pk}}_0) = 0$, $\mathsf{cred}_I = \perp$. Define $\mathsf{nym}'_0 = \breve{\mathsf{pk}}_0$, $\tilde{\sigma}_1 = \sigma_1$.
   Compute $\sigma_1 \leftarrow \mathsf{Sign}(\breve{\mathsf{sk}}_0, \mathsf{nym}_1)$.
   Output $\mathsf{cred}_R = (\mathsf{nym}_1, \sigma_1)$ and send it to the receiver.
2. If $L_I(\breve{\mathsf{pk}}_0) \neq 0$, $\mathsf{cred}_I = \{(\mathsf{nym}_1, \ldots, \mathsf{nym}_{L_I}), (\sigma_1, \ldots, \sigma_{L_I})\}$.
   Pick random $(\rho_1, \ldots, \rho_{L_I}) \leftarrow \mathsf{sample}_\rho^{L_I}$.
   If $L_I(\breve{\mathsf{pk}}_0) = 1$, $\mathsf{nym}'_0 = \breve{\mathsf{pk}}_0$, $\tilde{\sigma}_1 = \sigma_1$ as above.
   Compute $(\mathsf{nym}'_1, \sigma'_1) \leftarrow \mathsf{ChangeRep}(\mathsf{nym}'_0, \mathsf{nym}_1, \tilde{\sigma}_1, \rho_1)$.
   If $L_I(\breve{\mathsf{pk}}_0) > 1$, for $2 \leq i \leq L_I$,
   Compute $\tilde{\sigma}_i \leftarrow \mathsf{ConvertSig}(\mathsf{nym}_{i-1}, \mathsf{nym}_i, \sigma_i, \rho_{i-1})$.
   Then, compute $(\mathsf{nym}'_i, \sigma'_i) \leftarrow \mathsf{ChangeRep}(\mathsf{nym}'_{i-1}, \mathsf{nym}_i, \tilde{\sigma}_i, \rho_i)$.
   Finally, output $(\mathsf{nym}'_1, \ldots, \mathsf{nym}'_{L_I})$ and $(\sigma'_1, \ldots, \sigma'_{L_I})$.
   Compute $\sigma_{L_I+1} \leftarrow \mathsf{Sign}(\mathsf{sk}_{L_I}, \mathsf{nym}_R)$.
   Send the randomized chain as well as σ_{L_I+1} to the receiver.

$\mathsf{Receive}$: Store the resulting credential chain cred_R : $\{(\mathsf{nym}'_1, \ldots, \mathsf{nym}'_{L_I}, \mathsf{nym}_R), (\sigma'_1, \ldots, \sigma'_{L_I}, \sigma_{L_I+1})\}$.

Proof of possession of a credential:

$[\mathsf{CredProve}(L_P(\breve{\mathsf{pk}}_0), \breve{\mathsf{pk}}_0, \mathsf{sk}_P, \mathsf{nym}_P, \mathsf{aux}_P, \mathsf{cred}_P) \leftrightarrow \mathsf{CredVerify}(params, L_P(\breve{\mathsf{pk}}_0), \breve{\mathsf{pk}}_0, \mathsf{nym}_P)] \to$ output (0 or 1)

CredProve: Randomize the credential chain $\mathsf{cred}_P = \{(\mathsf{nym}_1, \ldots, \mathsf{nym}_{L_P}), (\sigma_1, \ldots, \sigma_{L_P})\}$ as above and send the chain to the verifier. The prover then proves knowledge of the secret key that corresponds to the last pseudonym, nym'_{L_P}, on the randomized chain.
CredVerify: If $\mathsf{cred}_P \neq \bot$, output 1; else, output 0.

Theorem 5 *The construction presented in Section 5 is correct, unforgeable, and anonymous in the single-authority security game,*

Unforgeability follows from unforgeability of mercurial signatures, while anon ymity follows from class- and origin-hiding. The proof is in Appendix D.

Efficiency analysis. Let us see what happens when this scheme is instantiated with our mercurial signature construction presented in Section 3. First of all, we need to establish an appropriate value for the length parameter ℓ. It is easy to see that $\ell = 2$ is good enough for our purposes, so that public keys and messages consist of two group elements each ($\ell = 1$ is not good enough because then all messages and all public keys are equivalent.) A certification chain of length L will then have two group elements per pseudonym on the chain and three group elements per signature (according to our construction). Therefore, it takes $5L$ group elements to represent a credential chain of length L.

Trust assumptions. Note that the only system-wide setup that needs to take place is the generation of the parameters for the system. When instantiated with mercurial signature from Section 3, this boils down to just the bilinear pairing setup. Even if the setup is carried out by a less-than-trustworthy party, it is unclear that this party can necessarily introduce trapdoors that would allow it to forge credentials or break anonymity. This is in contrast with previous constructions, in which the setup had to be carried out by a completely trustworthy process because trapdoors gleaned during this process would allow an adversary to completely break these systems (since these trapdoors allowed simulation and extraction, thus breaking both unforgeability and anonymity).

References

[BCC+09] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In *CRYPTO 2009*, v. 5677 of *LNCS*, p. 108–125. Springer, 2009.

[BHKS18] Michael Backes, Lucjan Hanzlik, Kamil Kluczniak, and Jonas Schneider. Signatures with flexible public key: A unified approach to privacy-preserving signatures (full version). `https://eprint.iacr.org/2018/191`.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, p. 136–145. IEEE Computer Society Press, 2001.

[CDD17] Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. Practical UC-secure delegatable credentials with attributes and their application to blockchain. In *ACM CCS 17*, pages 683–699. ACM Press, 2017.

[CDHK15] Jan Camenisch, Maria Dubovitskaya, Kristiyan Haralambiev, and Markulf Kohlweiss. Composable and modular anonymous credentials: Definitions and practical constructions. In *ASIACRYPT 2015, Part II*, v. 9453 of *LNCS*, p. 262–288. Springer, 2015.

[Cha86] David Chaum. Showing credentials without identification: Signatures transferred between unconditionally unlinkable pseudonyms. In Franz Pichler, editor, *EUROCRYPT'85*, v. 219 of *LNCS*, p. 241–244. Springer, 1986.

[CHK+05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *EUROCRYPT 2005*, v. 3494 of *LNCS*, p. 404–421. Springer, 2005.

[CKL+14] Jan Camenisch, Stephan Krenn, Anja Lehmann, Gert Læssøe Mikkelsen, Gregory Neven, and Michael Østergaard Pedersen. Formal treatment of privacy-enhancing credential systems. `http://eprint.iacr.org/2014/708`.

[CKLM13] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable signatures: Complex unary transformations and delegatable anonymous credentials. `http://eprint.iacr.org/2013/179`.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, v. 2045 of *LNCS*, p. 93–118. Springer, May 2001.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *CRYPTO 2004*, v. 3152 of *LNCS*, p. 56–72. Springer, August 2004.

[CL06] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In Cynthia Dwork, editor, *CRYPTO 2006*, v. 4117 of *LNCS*, p. 78–96. Springer, August 2006.

[Dam00] Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In Bart Preneel, editor, *EUROCRYPT 2000*, v. 1807 of *LNCS*, p. 418–430. Springer, May 2000.

[FHS14] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. `http://eprint.iacr.org/2014/944`.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, v. 4965 of *LNCS*, p. 415–432. Springer, April 2008.

[Lin03a] Yehuda Lindell. Bounded-concurrent secure two-party computation without setup assumptions. In *35th ACM STOC*, p. 683–692. ACM Press, June 2003.

[Lin03b] Yehuda Lindell. Brief announcement: impossibility results for concurrent secure two-party computation. In Elizabeth Borowsky and Sergio Rajsbaum, editors, *22nd ACM PODC*, page 200. ACM, July 2003.

[LRSW99] Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In Howard M. Heys and Carlisle M. Adams, editors, *SAC 1999*, v. 1758 of *LNCS*, p. 184–199. Springer, August 1999.

[Lys02] Anna Lysyanskaya. *Signature schemes and applications to cryptographic protocol design*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, September 2002.

[Swe97] Latanya Sweeney. Weaving technology and policy together to maintain confidentiality. *International Journal of Law, Medicine and Ethics*, 2,3(25):98–110, 1997.

A Construction of Mercurial Signatures in Additive Notation

Proofs of correctness, unforgeability, message and public key class-hiding of mercurial signatures are written in additive notation for clarity, so here we provide preliminaries on bilinear groups and the mercurial signature construction in additive notation.

A.1 Preliminaries on bilinear groups

Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be groups of prime order p, where $\mathbb{G}_1, \mathbb{G}_2$ are written additively and $\mathbb{G}_T$ is written multiplicatively. Let $P, \hat{P}$ be generators of $\mathbb{G}_1, \mathbb{G}_2$, respectively. A *bilinear pairing* is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ that satisfies:

Bilinearity: $\forall a, b \in \mathbb{Z}_p, \forall R \in \mathbb{G}_1, \forall S \in \mathbb{G}_2 : e(aR, bS) = e(R, S)^{ab}$.
Non-degeneracy: $e(P, \hat{P}) \neq 1$ (i.e. $e(P, \hat{P})$ generates $\mathbb{G}_T$).
Computability: There exists an efficient algorithm to compute e.

Bilinear groups can be classified into three types:

Type I (symmetric): $\mathbb{G}_1 = \mathbb{G}_2$.
Type II (asymmetric): $\mathbb{G}_1 \neq \mathbb{G}_2$, but there exists an efficiently computable homomorphism $\phi : \mathbb{G}_2 \to \mathbb{G}_1$ (none in the reverse direction).
Type III (asymmetric): $\mathbb{G}_1 \neq \mathbb{G}_2$, but there exists no efficiently computable homomorphism in either direction.

Let BGGen be a polynomial-time algorithm that, on input the security parameter 1^k, outputs descriptions of the groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, generators $P, \hat{P}$ of $\mathbb{G}_1, \mathbb{G}_2$, and the algorithm for the bilinear map e. Let BG denote all of these parameters together.

A.2 Mercurial signature construction in additive notation

Let $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ be as above. The message space for our mercurial signature scheme will consist of vectors of group elements from $\mathbb{G}_1^*$, where $\mathbb{G}_1^* = \mathbb{G}_1 \backslash \{0_{\mathbb{G}_1}\}$. The space of secret keys will consist of vectors of elements from $\mathbb{Z}_p^*$. The space of public keys, similar to the message space, will consist of vectors of group elements from $\mathbb{G}_2^*$. Once the prime p, $\mathbb{G}_1^*$, $\mathbb{G}_2^*$, and ℓ are well-defined, the equivalence relations of interest to us are as follows:

$$\mathcal{R}_M = \{(M, M') \in (\mathbb{G}_1^*)^\ell \times (\mathbb{G}_1^*)^\ell \mid \exists r \in \mathbb{Z}_p^* \text{ such that } M' = r \cdot M\}$$

$$\mathcal{R}_{\mathsf{sk}} = \{(\mathsf{sk}, \tilde{\mathsf{sk}}) \in (\mathbb{Z}_p^*)^\ell \times (\mathbb{Z}_p^*)^\ell \mid \exists r \in \mathbb{Z}_p^* \text{ such that } \tilde{\mathsf{sk}} = r \cdot \mathsf{sk}\}$$

$$\mathcal{R}_{\mathsf{pk}} = \{(\mathsf{pk}, \tilde{\mathsf{pk}}) \in (\mathbb{G}_2^*)^\ell \times (\mathbb{G}_2^*)^\ell \;\mid\; \exists r \in \mathbb{Z}_p^* \text{ such that } \tilde{\mathsf{pk}} = r \cdot \mathsf{pk}\}$$

Note that messages, secret keys, and public keys are restricted to vectors with nonzero entries. Without this restriction and the restriction that $r \neq 0$, the resulting relation would not be an equivalence one.

Here is the mercurial signature construction with message space $(\mathbb{G}_1^*)^\ell$ in additive notation.

$\mathsf{PPGen}(1^k) \to PP$: Compute $\mathsf{BG} \leftarrow \mathsf{BGGen}(1^k)$. Output $PP = \mathsf{BG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, \hat{P}, e)$. Note that now that BG is well-defined, the relations $\mathcal{R}_M$, $\mathcal{R}_{\mathsf{pk}}$, $\mathcal{R}_{\mathsf{sk}}$ are also well-defined. sample_ρ and sample_μ are the same algorithm, namely the one that samples a random element of $\mathbb{Z}_p^*$.

$\mathsf{KeyGen}(PP, \ell) \to (\mathsf{pk}, \mathsf{sk})$: For $1 \leq i \leq \ell$, pick $x_i \leftarrow \mathbb{Z}_p^*$ and set secret key $\mathsf{sk} = (x_1, \ldots, x_\ell)$. Compute public key $\mathsf{pk} = (\hat{X}_1, \ldots, \hat{X}_\ell)$, where $\hat{X}_i = x_i \hat{P}$ for $1 \leq i \leq \ell$. Output $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Sign}(\mathsf{sk}, M) \to \sigma$: On input $\mathsf{sk} = (x_1, \ldots, x_\ell)$ and $M = (M_1, \ldots, M_\ell) \in (\mathbb{G}_1^*)^\ell$, pick a random $y \leftarrow \mathbb{Z}_p^*$ and output $\sigma = (Z, Y, \hat{Y})$, where

$$Z \leftarrow y \sum_{i=1}^{\ell} x_i M_i \qquad Y \leftarrow \frac{1}{y} P \qquad \hat{Y} \leftarrow \frac{1}{y} \hat{P}$$

$\mathsf{Verify}(\mathsf{pk}, M, \sigma) \to 0/1$: On input $\mathsf{pk} = (\hat{X}_1, \ldots, \hat{X}_\ell)$, $M = (M_1, \ldots, M_\ell)$, and $\sigma = (Z, Y, \hat{Y})$, check whether

$$\prod_{i \in [\ell]} e(M_i, \hat{X}_i) = e(Z, \hat{Y}) \;\; \wedge \;\; e(Y, \hat{P}) = e(P, \hat{Y})$$

If the above holds, output 1; otherwise output 0.

$\mathsf{ConvertSK}(\mathsf{sk}, \rho) \to \tilde{\mathsf{sk}}$: On input $\mathsf{sk} = (x_1, \ldots, x_\ell)$ and a key converter $\rho \in \mathbb{Z}_p^*$, output the new secret key $\tilde{\mathsf{sk}} = \rho \cdot \mathsf{sk}$.

$\mathsf{ConvertPK}(\mathsf{pk}, \rho) \to \tilde{\mathsf{pk}}$: On input $\mathsf{pk} = (\hat{X}_1, \ldots, \hat{X}_\ell)$ and a key converter $\rho \in \mathbb{Z}_p^*$, output the new public key $\tilde{\mathsf{pk}} = \rho \cdot \mathsf{pk}$.

$\mathsf{ConvertSig}(\mathsf{pk}, M, \sigma, \rho) \to \tilde{\sigma}$: On input pk, message M, signature $\sigma = (Z, Y, \hat{Y})$, and key converter $\rho \in \mathbb{Z}_p^*$, sample $\psi \leftarrow \mathbb{Z}_p^*$. Output $\tilde{\sigma} = (\psi\rho Z, \frac{1}{\psi} Y, \frac{1}{\psi} \hat{Y})$.

$\mathsf{ChangeRep}(\mathsf{pk}, M, \sigma, \mu) \to (M', \sigma')$: On input pk, M, $\sigma = (Z, Y, \hat{Y})$, $\mu \in \mathbb{Z}_p^*$, sample $\psi \leftarrow \mathbb{Z}_p^*$. Compute $M' = \mu M$, $\sigma' = (\psi\mu Z, \frac{1}{\psi} Y, \frac{1}{\psi} \hat{Y})$. Output (M', σ').

B Proofs of correctness, unforgeability, message and public key class-hiding

B.1 Correctness of mercurial signatures

Correct verification and correct key conversion can be seen by inspection. We show correct signature conversion, and correct change of message representative is similar.

Proof. We wish to show that for all $M \in \mathcal{M}$, for all σ such that $\mathsf{Verify}(\mathsf{pk}, M, \sigma) = 1$, for all $\rho \in \mathsf{sample}_\rho$, for all $\tilde{\sigma} \in \mathsf{ConvertSig}(\mathsf{pk}, M, \sigma, \rho)$, $\mathsf{Verify}(\mathsf{ConvertPK}(\mathsf{pk}, \rho), M, \tilde{\sigma}) = 1$. We proceed as follows. $\mathsf{ConvertSig}$ returns $\tilde{\sigma} = (\psi\rho Z, \frac{1}{\psi}Y, \frac{1}{\psi}\hat{Y})$ and $\mathsf{ConvertPK}$ returns $\tilde{\mathsf{pk}} = \rho \cdot \mathsf{pk}$. The second verification equation, $e(Y, \hat{P}) = e(P, \hat{Y})$, holds because $e(Y, \hat{P}) = e(\frac{1}{y}P, \hat{P}) = e(P, \frac{1}{y}\hat{P}) = e(P, \hat{Y})$. Plugging in $\tilde{\mathsf{pk}} \leftarrow \rho \cdot \mathsf{pk}$ and $\tilde{\sigma} = (\psi\rho Z, \frac{1}{\psi}Y, \frac{1}{\psi}\hat{Y})$, we see that the first verification equation holds as well:

$$e(\psi\rho Z, \frac{1}{\psi}\hat{Y}) = e(\rho Z, \hat{Y})^{\psi \cdot \frac{1}{\psi}} = e(\rho y \sum_{i \in [\ell]} x_i M_i, \frac{1}{y}\hat{P}) = e(\sum_{i \in [\ell]} \rho x_i M_i, \hat{P})^{y \cdot \frac{1}{y}}$$
$$= \prod_{i \in [\ell]} e(\rho x_i M_i, \hat{P}) = \prod_{i \in [\ell]} e(M_i, \rho \hat{X}_i)$$

B.2 Unforgeability of mercurial signatures

We now prove unforgeability for our construction of mercurial signatures. We require the following definition of unforgeability from [FHS14].

Definition 8 (EUF-CMA for SPS-EQ [FHS14]) *A structure-preserving signature scheme for equivalence relation $\mathcal{R}$ over $\mathbb{G}_i$ is existentially unforgeable under adaptive chosen-message attacks if for all $\ell > 1$ and all PPT algorithms $\mathcal{A}$ having access to a signing oracle $\mathsf{Sign}(\mathsf{sk}, \cdot)$, there is a negligible function ν such that:*

$$\Pr[\mathsf{BG} \leftarrow \mathsf{BGGen}(1^k), (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{BG}, \ell), (Q, M^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot)}(\mathsf{pk}) :$$
$$\forall M \in Q, [M^*]_{\mathcal{R}_M} \neq [M]_{\mathcal{R}_M} \ \wedge \ \mathsf{Verify}(M^*, \sigma^*, \mathsf{pk}) = 1] \leq \nu(k)$$

where Q is the set of queries that $\mathcal{A}$ has issued to the signing oracle.

In the generic group model, an adversary has access to encodings of group elements from $\mathbb{G}_1^*, \mathbb{G}_2^*$, and $\mathbb{G}_T$. $\mathcal{A}$ can query the respective group oracles for tests of group membership or equality, group operations, or bilinear pairings. If $\mathcal{A}$ requests a group operation or pairing on (encodings of) two group elements, $\mathcal{A}$ receives the encoding of the result.

We wish to show that no generic PPT adversary with access to a signing oracle can forge a mercurial signature with greater than negligible probability. We suppose such an adversary exists and arrive at a contradiction.

Proof. (of Theorem 2.) Suppose there exists a PPT algorithm $\mathcal{A}$ that can break the unforgeability of a mercurial signature scheme; that is, suppose $\mathcal{A}$ is able to produce a forgery $(M^*, \sigma^*, \mathsf{pk}^*)$ that satisfies the following conditions with non-negligible probability:

$$\forall M \in Q, [M^*]_{\mathcal{R}_M} \neq [M]_{\mathcal{R}_M} \ \wedge \ [\mathsf{pk}^*]_{\mathcal{R}_{\mathsf{pk}}} = [\mathsf{pk}]_{\mathcal{R}_{\mathsf{pk}}} \ \wedge \ \mathsf{Verify}(M^*, \sigma^*, \mathsf{pk}^*) = 1$$

where Q is the set of queries that $\mathcal{A}$ has issued to the signing oracle. The fact that pk^* belongs to the same equivalence class as pk implies there exists some $\alpha \in \mathbb{Z}_p^*$ such that $\mathsf{pk}^* = \alpha\mathsf{pk}$. We construct a reduction from $\mathcal{B}$ to $\mathcal{A}$, where $\mathcal{B}$ is a PPT algorithm that can break a secure SPS–EQ scheme using $\mathcal{A}$.

The challenger in the SPS–EQ unforgeability game for $\mathcal{B}$ chooses values $(x_i)_{i\in[\ell]} \leftarrow (\mathbb{Z}_p^*)^\ell$, sets $\mathsf{pk} = (\hat{X}_i)_{i\in[\ell]} = (x_i\hat{P})_{i\in[\ell]}$, and forwards pk to $\mathcal{B}$. $\mathcal{B}$ operates as follows:

On input pk,

1. $\mathcal{B}$ forwards pk to $\mathcal{A}$ and runs $\mathcal{A}(\mathsf{pk})$. $\mathcal{B}$ forwards $\mathcal{A}$'s group operation and signature queries to the respective oracles and forwards the results to $\mathcal{A}$.

2. $\mathcal{B}$ obtains $\mathcal{A}$'s forgery $(\mathsf{pk}^*, M^*, \sigma^*)$.

3. If, via this process, it is possible for $\mathcal{B}$ to obtain α, $\mathcal{B}$ outputs $(\mathsf{pk}, \alpha M^*, \sigma^*)$ as his forgery; else, $\mathcal{B}$ outputs 0.

Now, let us analyze this reduction.

Claim 1 *If* $[\mathsf{pk}^*]_{\mathcal{R}_{\mathsf{pk}}} = [\mathsf{pk}]_{\mathcal{R}_{\mathsf{pk}}}$, *then the generic group model reduction* $\mathcal{B}$ *can obtain* $\alpha \in \mathbb{Z}_p^*$ *such that* $\mathsf{pk}^* = \alpha\mathsf{pk}$.

Proof. Initially, before any queries are made, the elements of $\mathbb{G}_2^*$ that $\mathcal{A}$ has seen are $\hat{P}$ and $(\hat{X}_i)_{i\in[\ell]} = (x_i\hat{P})_{i\in[\ell]}$. After q signature queries, $\mathcal{A}$ has seen the additional elements $(\hat{Y}_j)_{j\in[q]} \in \mathbb{G}_2^*$. Since $\mathcal{A}$ is a generic forger, his forged public key $\mathsf{pk}^* = (\hat{X}_1^*, \hat{X}_2^*, ..., \hat{X}_\ell^*)$ must be computed as a linear combination of these elements:

$$\hat{X}_i^* = \pi_{x^*,i}\hat{P} + \sum_{j\in[\ell]} \chi_{x^*,i,j}\hat{X}_j + \sum_{k\in[q]} \psi_{x^*,i,k}\hat{Y}_k$$

for some $\pi_{x^*,i}, \chi_{x^*,i,j}, \psi_{x^*,i,k} \in \mathbb{Z}_p$ for $k \in [q]$ and $j \in [\ell]$. Taking discrete logarithms base $\hat{P}$, we get:

$$x_i^* = \pi_{x^*,i} + \sum_{j\in[\ell]} \chi_{x^*,i,j}x_j + \sum_{k\in[q]} \psi_{x^*,i,k}\frac{1}{y_k}$$

This is a multivariate Laurent polynomial of total degree $O(q)$ in $x_1, ..., x_\ell, y_1, \ldots, y_q$. Suppose that, despite seeing $\mathcal{A}$'s queries to the group and signing oracles and their results, there exists some $n \in [\ell]$ for which $\mathcal{B}$ cannot obtain $\alpha \in \mathbb{Z}_p^*$ such that $x_n^* = \alpha x_n$. By the Schwartz-Zippel lemma, the probability that that the two formally different polynomials x_n^* and αx_n collide by evaluating to the same value (or, equivalently, that the difference polynomial evaluates to zero) is $O(\frac{q}{p})$, which is negligible.

By Claim 1, if $\mathcal{A}$ succeeds in producing a forgery, $\mathcal{B}$ obtains α and can produce $(\mathsf{pk}, \alpha M^*, \sigma^*)$, which has the following relationship to $\mathcal{A}$'s forgery $(\mathsf{pk}^*, M^*, \sigma^*)$:

$$\sigma^* = (Z^*, Y^*, \hat{Y}^*) = (y\sum_{i\in[\ell]} x_i^* M_i^*, Y^*, \hat{Y}^*) = (y\sum_{i\in[\ell]} x_i \alpha M_i^*, Y^*, \hat{Y}^*)$$

Therefore, a signature on M^* under pk^* is identical to a signature on αM^* under pk. Moreover, since $[\alpha M^*]_{\mathcal{R}_M} = [M^*]_{\mathcal{R}_M}$, we have that:

$$\forall M \in Q,\ [\alpha M^*]_{\mathcal{R}_M} \neq [M]_{\mathcal{R}_M} \ \ \wedge\ \mathsf{Verify}(\mathsf{pk}, \alpha M^*, \sigma^*) = 1$$

with non-negligible probability. Thus, $\mathcal{B}$ has produced a successful forgery against the SPS–EQ scheme in the generic group model, contradicting its proven security in that model.

B.3 Message and public key class-hiding of mercurial signatures

Proof. (of Theorem 3.) In [FHS14], the following result relating message class-hiding to the decisional Diffie-Hellman (DDH) assumption was shown.

Proposition 1 *Let $\ell > 1$. Then $(\mathbb{G}_i^*)^\ell$ is a message-hiding space if and only if the DDH assumption holds in $\mathbb{G}_i$.*

Now, we prove public key class-hiding.

Consider the following games involving a challenger in the generic group model. Without loss of generality, the message space is $(\mathbb{G}_1^*)^\ell$, and the public key space is $(\mathbb{G}_2^*)^\ell$. The adversary's bit $b \in \{0,1\}$ is used to distinguish between public keys pk_1 and pk_2 in the following argument.

Game 0. The generic group challenger computes the public parameters PP, which include a description of the bilinear group BG and generators $P, \hat{P}$ of $\mathbb{G}_1^*, \mathbb{G}_2^*$, respectively. The challenger also computes two independent public keys pk_1 and pk_2 as follows. He chooses $(x_i^{(1)})_{i\in[\ell]} \leftarrow (\mathbb{Z}_p^*)^\ell$ and computes $\mathsf{pk}_1 \leftarrow (\hat{X}_i^{(1)})_{i\in[\ell]} = (x_i^{(1)}\hat{P})_{i\in[\ell]}$. Similarly, he chooses $(x_i^{(2)})_{i\in[\ell]} \leftarrow (\mathbb{Z}_p^*)^\ell$ and computes $\mathsf{pk}_2 \leftarrow (\hat{X}_i^{(2)})_{i\in[\ell]} = (x_i^{(2)}\hat{P})_{i\in[\ell]}$. He then sends PP and "handles" for pk_1 and pk_2 to a generic adversary $\mathcal{A}$. The adversary can query the respective group oracles for tests of group membership or equality, group operations or bilinear pairings and can also query the signing oracle. In response, the challenger computes the discrete logarithms of the handles given to him by $\mathcal{A}$ and computes the desired operation or signature. The challenger keeps a table of handles he has already computed. If the results of $\mathcal{A}$'s query are new group elements, he forms new handles in the table and returns the new handles to $\mathcal{A}$; else, he returns the appropriate existing handles from the table.

Game 1. The challenger computes the public parameters PP. He then sends PP and handles for pk_1 and pk_2 to $\mathcal{A}$, where pk_1 and pk_2 are again independent. The adversary makes queries to the generic group oracles and signing oracle, and the challenger computes the formal multivariate Laurent polynomials associated with each query (we will see this in detail below.) For new polynomials, the challenger forms new corresponding handles in the table and returns the new handles to $\mathcal{A}$; else, he returns the appropriate existing handles from the table.

Game 2. The challenger computes the public parameters PP. In this game, pk_2 is in the same equivalence class as pk_1, so there exists some $\alpha \in \mathbb{Z}_p^*$ such that $\mathsf{pk}_2 = \alpha\mathsf{pk}_1$. The challenger sends PP and handles for pk_1 and pk_2 to $\mathcal{A}$. The

adversary makes queries to the generic group oracles and signing oracle. As in Game 1, the challenger computes the formal multivariate Laurent polynomials associated with each query, creates new handles in its table for new polynomials, and returns new or existing handles to $\mathcal{A}$ appropriately. Note that the formal multivariate Laurent polynomials will now include the additional variable α.

Game 3. The challenger computes the public parameters PP. In this game, pk_2 is again in the same equivalence class as pk_1, and the challenger computes them as follows. He chooses $(x_i^{(1)})_{i\in[\ell]} \leftarrow (\mathbb{Z}_p^*)^\ell$ and computes $\mathsf{pk}_1 \leftarrow (\hat{X}_i^{(1)})_{i\in[\ell]} = (x_i^{(1)}\hat{P})_{i\in[\ell]}$. He then chooses $\alpha \in \mathbb{Z}_p^*$ and computes $\mathsf{pk}_2 = (x_i^{(2)}\hat{P})_{i\in[\ell]} = \alpha\mathsf{pk}_1 = (\alpha x_i^{(1)}\hat{P})_{i\in[\ell]}$. The challenger sends PP and "handles" for pk_1 and pk_2 to $\mathcal{A}$. The adversary makes queries to the generic group oracles and signing oracle. The challenger computes the discrete logarithms of the handles it receives from $\mathcal{A}$, performs the appropriate computations, creates new handles in its table for new group elements if necessary, and returns new or existing handles to $\mathcal{A}$ appropriately.

In order to achieve class-hiding for public keys, $\mathcal{A}$'s view in each of these games must be the same.

Claim 2 *A generic adversary's view in Game 1 is the same as it is in Game 2.*

This holds if in each of the groups, $\mathbb{G}_1^*, \mathbb{G}_2^*$ and $\mathbb{G}_T$, the adversary's view is the same in both games.

Proof. **Step 1.** We first consider computations the challenger carries out in $\mathbb{G}_2^*$.

Initially, before any queries are made, the elements of $\mathbb{G}_2^*$ the adversary has seen are $\hat{P}, (\hat{X}_i^{(1)})_{i\in[\ell]} = (x_i^{(1)}\hat{P})_{i\in[\ell]}$, and $(\hat{X}_i^{(2)})_{i\in[\ell]} = (x_i^{(2)}\hat{P})_{i\in[\ell]}$. When $\mathcal{A}$ makes a query to a generic group oracle, the challenger computes a formal multivariate Laurent polynomial in the variables $x_1^{(1)}, x_2^{(1)}, \ldots, x_\ell^{(1)}, x_1^{(2)}, x_2^{(2)}, \ldots, x_\ell^{(2)}$. Suppose $\mathcal{A}$ submits a query in Game 1, in which pk_1 and pk_2 are unrelated, the result of which will also be an element of $\mathbb{G}_2^*$. The handle computed by the challenger will be a linear combination:

$$\hat{H}_i = \pi_{h,i}\hat{P} + \sum_{j\in[\ell]} \chi_{h,i,j}^{(1)}\hat{X}_j^{(1)} + \sum_{j\in[\ell]} \chi_{h,i,j}^{(2)}\hat{X}_j^{(2)} \tag{1}$$

for some $\pi_{h,i}, \chi_{h,i,j}^{(1)}, \chi_{h,i,j}^{(2)} \in \mathbb{Z}_p$ for $j \in [\ell]$. Taking discrete logarithms base $\hat{P}$, we get:

$$h_i = \pi_{h,i} + \sum_{j\in[\ell]} \chi_{h,i,j}^{(1)}x_j^{(1)} + \sum_{j\in[\ell]} \chi_{h,i,j}^{(2)}x_j^{(2)} \tag{2}$$

This is a formal multivariate Laurent polynomial in the variables $x_1^{(1)}, x_2^{(1)}, \ldots, x_\ell^{(1)}, x_1^{(2)}, x_2^{(2)}, \ldots, x_\ell^{(2)}$. If $\mathcal{A}$ submits the same query in Game 2, in which pk_1 and pk_2 are related by $\mathsf{pk}_2 = \alpha\mathsf{pk}_1$ for some $\alpha \in \mathbb{Z}_p^*$, the handle computed by the challenger will be a linear combination:

$$\hat{H}_i^{(\alpha)} = \pi_{h,i}\hat{P} + \sum_{j\in[\ell]} \chi_{h,i,j}^{(1)} \hat{X}_j^{(1)} + \sum_{j\in[\ell]} \chi_{h,i,j}^{(2)} \alpha \hat{X}_j^{(1)} \tag{3}$$

for the same $\pi_{h,i}, \chi_{h,i,j}^{(1)}, \chi_{h,i,j}^{(2)} \in \mathbb{Z}_p$ for $j \in [\ell]$ as in Equation (1). Taking discrete logarithms base $\hat{P}$, we get:

$$h_i^{(\alpha)} = \pi_{h,i} + \sum_{j\in[\ell]} \chi_{h,i,j}^{(1)} x_j^{(1)} + \sum_{j\in[\ell]} \chi_{h,i,j}^{(2)} \alpha x_j^{(1)} \tag{4}$$

This is a formal multivariate Laurent polynomial in the variables $x_1^{(1)}, x_2^{(1)}, \ldots, x_\ell^{(1)}$, and α.

Remark 1 *The formal Laurent polynomials in Equations (2) and (4) have identical coefficients, so there is a one-to-one correspondence between their monomials; however, they are formally different Laurent polynomials because α is a variable. We now show that two generic group queries to $\mathbb{G}_2^*$ in Game 1 result in distinct formal polynomials if and only if the same two queries in Game 2 result in distinct polynomials.*

Consider a second query in Game 1, resulting in an element of $\mathbb{G}_2^*$ corresponding to the formal Laurent polynomial:

$$\tilde{h}_i = \tilde{\pi}_{h,i} + \sum_{j\in[\ell]} \tilde{\chi}_{h,i,j}^{(1)} x_j^{(1)} + \sum_{j\in[\ell]} \tilde{\chi}_{h,i,j}^{(2)} x_j^{(2)} \tag{5}$$

Consider the same query in Game 2, which corresponds to the formal Laurent polynomial:

$$\tilde{h}_i^{(\alpha)} = \tilde{\pi}_{h,i} + \sum_{j\in[\ell]} \tilde{\chi}_{h,i,j}^{(1)} x_j^{(1)} + \sum_{j\in[\ell]} \tilde{\chi}_{h,i,j}^{(2)} \alpha x_j^{(1)} \tag{6}$$

Suppose h_i and $\tilde{h}_i$ resulting from the two queries in Game 1 are formally different polynomials. Then there exists some coefficient where their monomials differ. Because of the one-to-one correspondence between h_i and $h_i^{(\alpha)}$, they differ at the corresponding monomial. Because of the one-to-one correspondence between $\tilde{h}_i$ and $\tilde{h}_i^{(\alpha)}$, they differ at the same corresponding monomial too. Thus, $h_i^{(\alpha)}$ and $\tilde{h}_i^{(\alpha)}$ are formally different polynomials in Game 2. The other direction is similar.

Now, consider $\mathcal{A}$'s first query to the signing oracle. $\mathcal{A}$ chooses a public key, either pk_1 or pk_2, and a message M to be signed under that public key. The challenger returns the signature $(Z, Y, \hat{Y})$ on M under the desired public key. This introduces the element $\hat{Y} \in \mathbb{G}_2^*$, so a handle computed by the challenger in Game 1 will be a linear combination:

$$h_i = \pi_{h,i} + \sum_{j\in[\ell]} \chi_{h,i,j}^{(1)} x_j^{(1)} + \sum_{j\in[\ell]} \chi_{h,i,j}^{(2)} x_j^{(2)} + \psi_{h,i} \frac{1}{y} \tag{7}$$

where $\pi_{h,i}, \chi^{(1)}_{h,i,j}, \chi^{(2)}_{h,i,j}, \psi_{h,i} \in \mathbb{Z}_p$ for $j \in [\ell]$. This is a formal multivariate Laurent polynomial in $x^{(1)}_1, ..., x^{(1)}_\ell$, $x^{(2)}_1, ..., x^{(2)}_\ell$, and y. The same query in Game 2 corresponds to the linear combination:

$$h^{(\alpha)}_i = \pi_{h,i} + \sum_{j\in[\ell]} \chi^{(1)}_{h,i,j} x^{(1)}_j + \sum_{j\in[\ell]} \chi^{(2)}_{h,i,j} \alpha x^{(1)}_j + \psi_{h,i} \frac{1}{y} \tag{8}$$

for the same $\pi_{h,i}, \chi^{(1)}_{h,i,j}, \chi^{(2)}_{h,i,j}, \psi_{h,i} \in \mathbb{Z}_p$ for $j \in [\ell]$. Since the $1/y$ term is identical in h_i and $h^{(\alpha)}_i$, the one-to-one correspondence still holds. Thus, the statement that two queries to $\mathbb{G}^*_2$ in Game 1 result in distinct formal polynomials if and only if the same two queries in Game 2 result in distinct polynomials still holds with the addition of one signature query.

As $\mathcal{A}$ makes more signature queries, the elements in $\mathbb{G}^*_2$ are $\hat{P}, (\hat{X}^{(1)}_i)_{i\in[\ell]}$, $(\hat{X}^{(2)}_i)_{i\in[\ell]}$ and $(\hat{Y}_j)_{j\in[q]}$, where q is the number of queries $\mathcal{A}$ has issued to the signing oracle. Thus, any handle in $\mathbb{G}^*_2$ must be computed as a linear combination of these elements:

$$\hat{H}_i = \pi_{h,i}\hat{P} + \sum_{j\in[\ell]} \chi^{(1)}_{h,i,j} \hat{X}^{(1)}_j + \sum_{j\in[\ell]} \chi^{(2)}_{h,i,j} \hat{X}^{(2)}_j + \sum_{k\in[q]} \psi_{h,i,k} \hat{Y}_k \tag{9}$$

for some $\pi_{h,i}, \chi^{(1)}_{h,i,j}, \chi^{(2)}_{h,i,j}, \psi_{h,i,k} \in \mathbb{Z}_p$ for $j \in [\ell]$ and $k \in [q]$. Taking discrete logarithms base $\hat{P}$, we get:

$$h_i = \pi_{h,i} + \sum_{j\in[\ell]} \chi^{(1)}_{h,i,j} x^{(1)}_j + \sum_{j\in[\ell]} \chi^{(2)}_{h,i,j} x^{(2)}_j + \sum_{k\in[q]} \psi_{h,i,k} \frac{1}{y_k} \tag{10}$$

This is a formal Laurent polynomial of total degree $O(q)$ in $x^{(1)}_1, ..., x^{(1)}_\ell, x^{(2)}_1, ..., x^{(2)}_\ell, y_1, ...y_q$. The same query in Game 2 corresponds to the linear combination:

$$h^{(\alpha)}_i = \pi_{h,i} + \sum_{j\in[\ell]} \chi^{(1)}_{h,i,j} x^{(1)}_j + \sum_{j\in[\ell]} \chi^{(2)}_{h,i,j} \alpha x^{(1)}_j + \sum_{k\in[q]} \psi_{h,i,k} \frac{1}{y_k} \tag{11}$$

Since the $1/y_k$ terms are identical in h_i and $h^{(\alpha)}_i$, the one-to-one correspondence still holds. Thus, the statement that two queries to $\mathbb{G}^*_2$ in Game 1 result in distinct formal polynomials if and only if the same two queries in Game 2 result in distinct polynomials still holds with the addition of q signature queries. This captures precisely the notion that $\mathcal{A}$'s view for queries in $\mathbb{G}^*_2$ is the same in both games.

Step 2. We now consider computations the challenger carries out in $\mathbb{G}^*_1$.

Before any signature queries, the only element of $\mathbb{G}^*_1$ that $\mathcal{A}$ has seen is P. Therefore, she must form her first message M_1 to be signed as $M_1 = (m_{1,i}P)_{i\in[\ell]}$, where $m_{1,i} = \pi_{m,1,i}$ for some $\pi_{m,1,i} \in \mathbb{Z}_p$. $\mathcal{A}$ chooses the public key, either pk_1 or pk_2, under which M_1 is to be signed. We recall that $\mathsf{pk}_1 = (x^{(1)}_i P)_{i\in[\ell]}$ for some $x^{(1)}_i \in \mathbb{Z}^*_p$, and $\mathsf{pk}_2 = (x^{(2)}_i P)_{i\in[\ell]}$ for some $x^{(2)}_i \in \mathbb{Z}^*_p$. The challenger returns the

signature $(Z_1^{(1)}, Y_1^{(1)}, \hat{Y}_1^{(1)})$ on M_1 under pk_1 or the signature $(Z_1^{(2)}, Y_1^{(2)}, \hat{Y}_1^{(2)})$ on M_1 under pk_2. In each case, this introduces two new elements in $\mathbb{G}_1^*$: $(Z_1^{(1)}, Y_1^{(1)})$ or $(Z_1^{(2)}, Y_1^{(2)})$. The discrete logarithms base P of $Y_1^{(1)}$ and $Y_1^{(2)}$ are $1/y_1^{(1)}$ and $1/y_2^{(2)}$, respectively. The discrete logarithms base P of $Z_1^{(1)}$ and $Z_1^{(2)}$ are as follows:

$$z_1^{(1)} = y_1^{(1)} \sum_{i \in [\ell]} x_i^{(1)} m_{1,i} = y_1^{(1)} \sum_{i \in [\ell]} x_i^{(1)} \pi_{m,1,i} \tag{12}$$

$$z_1^{(2)} = y_1^{(2)} \sum_{i \in [\ell]} x_i^{(2)} m_{1,i} = y_1^{(2)} \sum_{i \in [\ell]} x_i^{(2)} \pi_{m,1,i} \tag{13}$$

Equation (12) is a formal Laurent polynomial in $x_1^{(1)}, x_2^{(1)}, \ldots, x_\ell^{(1)}$ and $y_1^{(1)}$, and Equation (13) is a formal Laurent polynomial in $x_1^{(2)}, x_2^{(2)}, \ldots, x_\ell^{(2)}$ and $y_1^{(2)}$.

$\mathcal{A}$ makes a second signature query by choosing a message $M_2 = (m_{2,i}P)_{i \in [\ell]}$ and a public key, pk_1 or pk_2, under which M_2 is to be signed. For all $i, m_{2,i}$ is now a linear combination of elements in $\mathbb{G}_1^*$ that $\mathcal{A}$ has seen as result of the first signature query, so $m_{2,i}$ takes one of the following two forms according to whether M_1 was signed under pk_1 or pk_2:

$$m_{2,i}^{(1)} = \pi_{m,2,i}^{(1)} + \rho_{m,2,i,1}^{(1)} z_1^{(1)} + \psi_{m,2,i,1}^{(1)} \frac{1}{y_1^{(1)}} \tag{14}$$

$$m_{2,i}^{(2)} = \pi_{m,2,i}^{(2)} + \rho_{m,2,i,1}^{(2)} z_1^{(2)} + \psi_{m,2,i,1}^{(2)} \frac{1}{y_1^{(2)}} \tag{15}$$

for some $\pi_{m,2,i}^{(1)}, \rho_{m,2,i,1}^{(1)}, \psi_{m,2,i,1}^{(1)}, \pi_{m,2,i}^{(2)}, \rho_{m,2,i,1}^{(2)}, \psi_{m,2,i,1}^{(2)}, \in \mathbb{Z}_p$. Equation (14) corresponds to M_1 having been signed under pk_1, and Equation (15) corresponds to M_1 having been signed under pk_2. Given that M_2 can take one of the two forms above, there are four possibilites for signatures output by the challenger:

$$(Z_2^{(1,1)}, Y_2^{(1,1)}, \hat{Y}_2^{(1,1)})\ (Z_2^{(1,2)}, Y_2^{(1,2)}, \hat{Y}_2^{(1,2)})$$
$$(Z_2^{(2,1)}, Y_2^{(2,1)}, \hat{Y}_2^{(2,1)})\ (Z_2^{(2,2)}, Y_2^{(2,2)}, \hat{Y}_2^{(2,2)})$$

The first element of an upper index denotes the public key under which M_1 is signed, and the second element denotes the public key under which M_2 is signed. Taking discrete logarithms base P, we have:

$$z_2^{(1,2)} = y_2^{(1,2)} \sum_{i \in [\ell]} x_i^{(2)} m_{2,i}^{(1)} \tag{16}$$

$$= y_2^{(1,2)} \sum_{i \in [\ell]} x_i^{(2)} (\pi_{m,2,i}^{(1)} + \rho_{m,2,i,1}^{(1)} z_1^{(1)} + \psi_{m,2,i,1}^{(1)} \frac{1}{y_1^{(1)}}) \tag{17}$$

$$= y_2^{(1,2)} \sum_{i \in [\ell]} x_i^{(2)} (\pi_{m,2,i}^{(1)} + \rho_{m,2,i,1}^{(1)} (y_1^{(1)} \sum_{j \in [\ell]} x_j^{(1)} \pi_{m,1,j}) + \psi_{m,2,i,1}^{(1)} \frac{1}{y_1^{(1)}}) \tag{18}$$

The equations for $z_2^{(2,1)}, z_2^{(1,1)}$ and $z_2^{(2,2)}$ are similar. This is a formal Laurent polynomial in $x_1^{(1)}, x_2^{(1)}, \ldots, x_\ell^{(1)}, x_1^{(2)}, x_2^{(2)}, \ldots, x_\ell^{(2)}$ and $y_1^{(1)}, y_2^{(1,2)}$. It can be seen by inspection that the monomials in $z_2^{(1,2)}$ take three forms: (1) $y_2^{(1,2)} x_i^{(2)}$; (2) $y_2^{(1,2)} x_i^{(2)} y_1^{(1)} x_j^{(1)}$; (3) $y_2^{(1,2)} x_i^{(2)} \frac{1}{y_1^{(1)}}$ for $1 \leq i, j \leq \ell$. We summarize the three forms of monomials in each case:

$$\begin{array}{llll}
z_2^{(1,2)} & : y_2^{(1,2)} x_i^{(2)} & y_2^{(1,2)} x_i^{(2)} y_1^{(1)} x_j^{(1)} & y_2^{(1,2)} x_i^{(2)} \frac{1}{y_1^{(1)}} \\
z_2^{(2,1)} & : y_2^{(2,1)} x_i^{(1)} & y_2^{(2,1)} x_i^{(1)} y_1^{(2)} x_j^{(2)} & y_2^{(2,1)} x_i^{(1)} \frac{1}{y_1^{(2)}} \\
z_2^{(1,1)} & : y_2^{(1,1)} x_i^{(1)} & y_2^{(1,1)} x_i^{(1)} y_1^{(1)} x_j^{(1)} & y_2^{(1,1)} x_i^{(1)} \frac{1}{y_1^{(1)}} \\
z_2^{(2,2)} & : y_2^{(2,2)} x_i^{(2)} & y_2^{(2,2)} x_i^{(2)} y_1^{(2)} x_j^{(2)} & y_2^{(2,2)} x_i^{(2)} \frac{1}{y_1^{(2)}}
\end{array}$$

Note that in each monomial in $z_2^{(\beta_1,\beta_2)}$, there is at least one y and one x, and there is the same number of y's and x's in the numerator. There is at most one y in the denominator, which does not cancel out.

Now, consider Game 2, in which $\mathsf{pk}_2 = \alpha \mathsf{pk}_1$ for some $\alpha \in \mathbb{Z}_p^*$. The same queries in Game 2 result in the following monomials:

$$\begin{array}{llll}
z_2^{(1,2)} : & y_2^{(1,2)} \alpha x_i^{(1)} & y_2^{(1,2)} \alpha x_i^{(1)} y_1^{(1)} x_j^{(1)} & y_2^{(1,2)} \alpha x_i^{(1)} \frac{1}{y_1^{(1)}} \\
z_2^{(2,1)} : & y_2^{(2,1)} x_i^{(1)} & y_2^{(2,1)} x_i^{(1)} y_1^{(2)} \alpha x_j^{(1)} & y_2^{(2,1)} x_i^{(1)} \frac{1}{y_1^{(2)}} \\
z_2^{(1,1)} : & y_2^{(1,1)} x_i^{(1)} & y_2^{(1,1)} x_i^{(1)} y_1^{(1)} x_j^{(1)} & y_2^{(1,1)} x_i^{(1)} \frac{1}{y_1^{(1)}} \\
z_2^{(2,2)} : & y_2^{(2,2)} \alpha x_i^{(1)} & y_2^{(2,2)} \alpha x_i^{(1)} y_1^{(2)} \alpha x_j^{(1)} & y_2^{(2,2)} \alpha x_i^{(1)} \frac{1}{y_1^{(2)}}
\end{array} \tag{19}$$

In Game 1, all of the monomials are distinct. We see that in Game 2, the monomials are also distinct. Thus, two formal Laurent polynomials in Game 1 formed after two signature queries are distinct if and only if they are distinct in Game 2. We now show that this property holds even after q signing queries.

Claim 3 *In Game 1, for all $n \geq 1$, the monomials that constitute $z_n^{(\beta_1,\beta_2,\ldots\beta_n)}$, where $\beta_i \in \{1,2\}$, have the form:*

$$\frac{1}{(y_s^{(\beta_1,\beta_2,\ldots\beta_s)})^b} \prod_{k\in[t]} y_{j_k}^{(\beta_1,\beta_2,\ldots\beta_{j_k})} \prod_{k\in[t]} x_{i_k}^{\beta_{i_k}} \tag{20}$$

where (1) $1 \leq t \leq n$, (2) for all k, $j_k \leq n$ and $s < j_k$, and (3) for all $k_1 \neq k_2, j_{k_1} \neq j_{k_2}$, (4) $j_t = n$, and (5) $b \in \{0,1\}$.

Proof. We proceed by induction on n. The cases when $n = 1$ and $n = 2$ have been shown above. Assume for all $k \in [n]$ that the monomials of $z_k^{(\beta_1,\beta_2,\ldots\beta_k)}$ are of the form in Equation (20). We have:

$$
\begin{aligned}
m_{n+1,i}^{(\beta_1,\beta_2,\ldots,\beta_n)} &= \pi_{m,n+1,i}^{(\beta_1,\beta_2,\ldots,\beta_n)} + \rho_{m,n+1,i,n}^{(\beta_1,\beta_2,\ldots,\beta_n)} z_n^{(\beta_1,\beta_2,\ldots,\beta_n)} \\
&\quad + \psi_{m,n+1,i,n}^{(\beta_1,\beta_2,\ldots,\beta_n)} \frac{1}{y_n^{(\beta_1,\beta_2,\ldots,\beta_n)}} \\
&\quad + \rho_{m,n+1,i,n-1}^{(\beta_1,\beta_2,\ldots,\beta_n)} z_{n-1}^{(\beta_1,\beta_2,\ldots,\beta_n-1)} + \psi_{m,n+1,i,n-1}^{(\beta_1,\beta_2,\ldots,\beta_n)} \frac{1}{y_{n-1}^{(\beta_1,\beta_2,\ldots,\beta_{n-1})}}
\end{aligned}
$$

and:

$$
\begin{aligned}
z_{n+1}^{(\beta_1,\beta_2,\ldots,\beta_{n+1})} &= \sum_{i\in[\ell]} y_{n+1}^{(\beta_1,\beta_2,\ldots,\beta_{n+1})} x_i^{\beta_{n+1}} \Bigg\{ \pi_{m,n+1,i}^{(\beta_1,\beta_2,\ldots,\beta_n)} \\
&\quad + \sum_{k\in[n]} \rho_{m,n+1,i,k}^{(\beta_1,\beta_2,\ldots,\beta_k)} z_k^{(\beta_1,\beta_2,\ldots,\beta_k)} + \sum_{k\in[n]} \psi_{m,n+1,i,k}^{(\beta_1,\beta_2,\ldots,\beta_k)} \tfrac{1}{y_k^{(\beta_1,\beta_2,\ldots,\beta_k)}} \Bigg\} \\
&= \sum_{i\in[\ell]} y_{n+1}^{(\beta_1,\beta_2,\ldots,\beta_{n+1})} x_i^{\beta_{n+1}} \pi_{m,n+1,i}^{(\beta_1,\beta_2,\ldots,\beta_n)} \\
&\quad + \sum_{i\in[\ell]} \sum_{k\in[n]} y_{n+1}^{(\beta_1,\beta_2,\ldots,\beta_{n+1})} x_i^{\beta_{n+1}} \rho_{m,n+1,i,k}^{(\beta_1,\beta_2,\ldots,\beta_k)} z_k^{(\beta_1,\beta_2,\ldots,\beta_k)} \\
&\quad + \sum_{i\in[\ell]} \sum_{k\in[n]} y_{n+1}^{(\beta_1,\beta_2,\ldots,\beta_{n+1})} x_i^{\beta_{n+1}} \psi_{m,n+1,i,k}^{(\beta_1,\beta_2,\ldots,\beta_k)} \tfrac{1}{y_k^{(\beta_1,\beta_2,\ldots,\beta_k)}}
\end{aligned} \tag{21}
$$

The monomials in the first and the last sum are in the form of Equation (20). Now, let's analyze the monomials in the middle sum. By the induction hypothesis, any monomial in z_k is of the form:

$$
\frac{1}{(y_s^{(\beta_1,\beta_2,\ldots\beta_s)})^b} \prod_{p\in[t]} y_{j_p}^{(\beta_1,\beta_2,\ldots\beta_{j_p})} \prod_{p\in[t]} x_{i_p}^{\beta_{i_p}}
$$

where $t \leq n, j_t = k$ and $s < j_p$ for all j_p as well as $j_p < k$, for all j_p with $p < t$ (which are all different). Each monomial of this form results in a monomial in the middle sum of the form:

$$
\begin{aligned}
&\frac{1}{(y_s^{(\beta_1,\beta_2,\ldots\beta_s)})^b} (y_{n+1}^{(\beta_1,\beta_2,\ldots\beta_{n+1})}) \prod_{p\in[t]} y_{j_p}^{(\beta_1,\beta_2,\ldots\beta_{j_p})})(x_i^{\beta_{n+1}} \prod_{p\in[t]} x_{i_p}^{\beta_{i_p}}) \\
&\qquad = \frac{1}{(y_s^{(\beta_1,\beta_2,\ldots\beta_s)})^b} \prod_{p\in[t']} y_{j_p}^{(\beta_1,\beta_2,\ldots\beta_{j_p})} \prod_{p\in[t']} x_{i_p}^{\beta_{i_p}}
\end{aligned} \tag{22}
$$

where $t' = t + 1 \leq n + 1, j_{t'} = n + 1, i_{t+1} = i$. Moreover, $t' \leq n + 1$, all j_p are still different and $\leq n$ and $s < j_p$ for all j_p, which completes the induction.

Claim 3 implies that in each monomial in $z_k^{(\beta_1,\beta_2,\ldots,\beta_k)}$, there is at least one y and one x, and there is the same number of y's and x's in the numerator. There is at most one y in the denominator, which does not cancel out; furthermore, Claim 3 implies the following corollary.

Corollary 1. *Any monomial can only occur in one unique* $z_n^{(\beta_1,\beta_2,\ldots,\beta_n)}$.

Proof. For a given monomial, let k be the maximal value such that the monomial contains $y_k^{(\beta_1,\beta_2,\ldots\beta_k)}$. Then the monomial is not contained in $z_n^{(\beta_1,\beta_2,\ldots,\beta_n)}$ with $n > k$; it would contradict maximality because $z_n^{(\beta_1,\beta_2,\ldots,\beta_n)}$ contains $y_n^{(\beta_1,\beta_2,\ldots,\beta_n)}$. The monomial is not contained in $z_n^{(\beta_1,\beta_2,\ldots\beta_n)}$) with $n < k$ either since all $y_j^{(\beta_1,\beta_2,\ldots\beta_j)}$ contained in $z_n^{(\beta_1,\beta_2,\ldots\beta_n)}$ are such that $j \leq n$. This would imply that $y_k^{(\beta_1,\beta_2,\ldots\beta_k)}$ is not contained in $z_n^{(\beta_1,\beta_2,\ldots\beta_n)}$, which is a contradiction.

We have shown distinctness among monomials within $z_n^{(\beta_1,\beta_2,\ldots,\beta_n)}$. The set of $z_n^{(\beta_1,\beta_2,\ldots,\beta_n)}$ are distinct from one other because each one represents a unique combination of $(\beta_1,\beta_2,\ldots,\beta_n) \in \{1,2\}^n$ with corresponding $y_n^{(\beta_1,\beta_2,\ldots,\beta_n)}$. Thus, all monomials in Game 1 are distinct.

In Game 2, in which $\mathsf{pk}_2 = \alpha\mathsf{pk}_1$, monomials containing $x_i^{(2)}$ become monomials containing $\alpha x_i^{(1)}$. It might seem that monomials involving products of the form $x_i^{(1)}x_i^{(2)}$ (ignoring the y's in these terms for a moment), which are distinct in Game 1, could no longer be distinct in Game 2 because $x_i^{(1)}x_i^{(2)} = x_i^{(1)}\alpha x_i^{(1)} = \alpha x_i^{(1)}x_i^{(1)} = x_i^{(2)}x_i^{(1)}$; however, because the y's for the terms on the left-hand side of the equation are distinct from the y's for the terms on the right-hand side, distinctness of monomials is preserved in Game 2. We see this explicitly in Equations (19) for the case of two signature queries. Thus, two formal Laurent polynomials in Game 1 formed after q signature queries are distinct if and only if they are distinct in Game 2. This captures precisely the notion that $\mathcal{A}$'s view for queries in $\mathbb{G}_1^*$ is the same in both games. Thus, $\mathcal{A}$'s view for queries in either $\mathbb{G}_1^*$ or $\mathbb{G}_2^*$ is the same in Game 1 and Game 2.

Step 3. We finally consider computations carried out by the challenger in $\mathbb{G}_T$. A formal Laurent polynomial in $\mathbb{G}_T$ arises as product of a formal polynomial in $\mathbb{G}_1^*$ and a formal polynomial in $\mathbb{G}_2^*$ via the bilinear map. Recall that a formal polynomial in $\mathbb{G}_2^*$ has monomials of the form $x_i^{(1)}, x_j^{(2)}$, and $1/y_k^{(\beta_1,\beta_2,\ldots\beta_k)}$, and a formal polynomial in $\mathbb{G}_1^*$ has monomials of the form $1/y_k^{(\beta_1,\beta_2,\ldots\beta_k)}$ and $z_n^{(\beta_1,\beta_2,\ldots\beta_n)}$. In Game 2, when every $x_j^{(2)}$ is replaced by $\alpha x_j^{(1)}$, the possibility of distinct monomials in Game 1 becoming indistinct in Game 2 could only occur for product monomials $x_j^{(2)}z_n^{(\beta_1,\beta_2,\ldots\beta_n)}$, which in Game 2 have the form (after q signature queries):

$$\alpha x_r^{(1)} \frac{1}{(y_s^{(\beta_1,\beta_2,\ldots\beta_s)})^b} \prod_{k\in[t]} y_{j_k}^{(\beta_1,\beta_2,\ldots\beta_{j_k})} \prod_{k\in[t]} x_{i_k}^{\beta_{i_k}} \tag{23}$$

Suppose there were a monomial that in Game 2 appeared the same as Equation (23), but with the α "absorbed" into the product of x's on the right, changing an $x_i^{(1)}$ to an $x_i^{(2)}$. This would still be distinct from Equation (23) because the $y_{j_k}^{(\beta_1,\beta_2,\ldots\beta_{j_k})}$'s would be distinct. We see this clearly in the example of two queries:

$$
\begin{array}{lllll}
z_2^{(1,2)}: & y_2^{(1,2)}\alpha x_i^{(1)}\alpha x_i^{(1)} & y_2^{(1,2)}\alpha x_i^{(1)}y_1^{(1)}x_j^{(1)}\alpha x_i^{(1)} & y_2^{(1,2)}\alpha x_i^{(1)}\frac{1}{y_1^{(1)}}\alpha x_i^{(1)} \\
z_2^{(2,1)}: & y_2^{(2,1)}x_i^{(1)}\alpha x_i^{(1)} & y_2^{(2,1)}x_i^{(1)}y_1^{(2)}\alpha x_j^{(1)}\alpha x_i^{(1)} & y_2^{(2,1)}x_i^{(1)}\frac{1}{y_1^{(2)}}\alpha x_i^{(1)} \\
z_2^{(1,1)}: & y_2^{(1,1)}x_i^{(1)}\alpha x_i^{(1)} & y_2^{(1,1)}x_i^{(1)}y_1^{(1)}x_j^{(1)}\alpha x_i^{(1)} & y_2^{(1,1)}x_i^{(1)}\frac{1}{y_1^{(1)}}\alpha x_i^{(1)} \\
z_2^{(2,2)}: & y_2^{(2,2)}\alpha x_i^{(1)}\alpha x_i^{(1)} & y_2^{(2,2)}\alpha x_i^{(1)}y_1^{(2)}\alpha x_j^{(1)}\alpha x_i^{(1)} & y_2^{(2,2)}\alpha x_i^{(1)}\frac{1}{y_1^{(2)}}\alpha x_i^{(1)}
\end{array}
$$

The first two monomials in the third column do not collide because of distinct y's. Thus, in $\mathbb{G}_T$, two formal Laurent polynomials in Game 1 formed after q signature queries are distinct if and only if they are distinct in Game 2.

Claim 4 *The adversary's view in Game 0 is the same as it is in Game 1.*

Proof. We first consider the computations the challenger carries out in $\mathbb{G}_2^*$. In both of these games, the public keys pk_1 and pk_2 are unrelated, so the formal polynomials are in the form of Equation (10). By the Schwartz-Zippel lemma, the probability that a formal polynomial in Game 1, in which the variables $x_1^{(1)}, ..., x_\ell^{(1)}, x_1^{(2)}, ..., x_\ell^{(2)}, y_1, ...y_q$ are given to $\mathcal{A}$ as handles, collides with a polynomial in Game 0, in which the handles correspond to the variables that were fixed at the beginning of the game, is negligible. The case for queries in $\mathbb{G}_1^*$ is similar.

Claim 5 *The adversary's view in Game 2 is the same as it is in Game 3.*

Proof. We first consider the computations the challenger carries out in $\mathbb{G}_2^*$. In both of these games, the public keys pk_1 and pk_2 are related by $\mathsf{pk}_2 = \alpha\mathsf{pk}_1$, so the formal polynomials are of the form:

$$h_i = \pi_{h,i} + \sum_{j\in[\ell]} \chi_{x,i,j}^{(1)} x_j^{(1)} + \sum_{j\in[\ell]} \chi_{x,i,j}^{(2)} \alpha x_j^{(1)} + \sum_{k\in[q]} \psi_{h,i,k}\frac{1}{y_k} \tag{24}$$

By the Schwartz-Zippel lemma, the probability that a formal polynomial in Game 2, in which the variables $x_1^{(1)}, ..., x_\ell^{(1)}, \alpha, y_1, ...y_q$ are given to $\mathcal{A}$ as handles, collides with a polynomial in Game 3, in which the handles correspond to the variables fixed at the beginning of the game, is negligible. The case for queries in $\mathbb{G}_1^*$ is similar.

This completes the proof of Theorem 3.

B.4 Origin-hiding of mercurial signatures

Proof.

Origin-hiding of ChangeRep**:** Let $\mathsf{pk}^*, M, \sigma = (Z, Y, \hat{Y})$ be such that $\mathsf{Verify}(\mathsf{pk}^*, M, \sigma) = 1$, where pk^* is possibly adversarially generated. For $\mu \in \mathbb{Z}_p^*$, $\mathsf{ChangeRep}(\mathsf{pk}^*, M, \sigma, \mu)$ outputs $(M', \sigma') = (\mu M, (\psi\mu Z, \frac{1}{\psi}Y, \frac{1}{\psi}\hat{Y}))$, which is a uniformly random element of $[M]_{\mathcal{R}_M}$ and a uniformly random element in the space of signatures $\hat{\sigma}$ satisfying $\mathsf{Verify}(\mathsf{pk}^*, \mu M, \hat{\sigma}) = 1$.

Origin-hiding of ConvertSig**:** Let $\mathsf{pk}^*, M, \sigma = (Z, Y, \hat{Y})$ be such that $\mathsf{Verify}(\mathsf{pk}^*, M, \sigma) = 1$, where pk^* is possibly adversarially generated. For $\rho \in \mathbb{Z}_p^*$, $\mathsf{ConvertSig}(\mathsf{pk}^*, M, \sigma, \rho)$ outputs $\tilde{\sigma} = (\psi\rho Z, \frac{1}{\psi}Y, \frac{1}{\psi}\hat{Y})$, which is a uniformly random element in the space of signatures $\hat{\sigma}$ satisfying $\mathsf{Verify}(\mathsf{ConvertPK}(\mathsf{pk}^*, \rho), M, \hat{\sigma}) = 1$, where $\mathsf{ConvertPK}(\mathsf{pk}^*, \rho) = \rho \cdot \mathsf{pk}^*$ is a uniformly random element of $[\mathsf{pk}^*]_{\mathcal{R}_{\mathsf{pk}}}$.

C Security Game for DAC

Below are formal descriptions of the oracles in the single-authority security game.

$\mathsf{AddHonestParty}(u)$: The adversary $\mathcal{A}$ invokes this oracle to create a new, honest node u. Let the current graph be $G(\breve{\mathsf{pk}}_0) = (V(\breve{\mathsf{pk}}_0), E(\breve{\mathsf{pk}}_0))$. If $u \in V(\breve{\mathsf{pk}}_0)$ (i.e. it is not a new node), abort. Else, the challenger $\mathcal{C}$ adds a new node u to $G(\breve{\mathsf{pk}}_0)$ (so $G(\breve{\mathsf{pk}}_0) := (V(\breve{\mathsf{pk}}_0) \cup \{u\}, E(\breve{\mathsf{pk}}_0))$). $\mathcal{C}$ then runs KeyGen to obtain $(\mathsf{pk}(u), \mathsf{sk}(u)) \leftarrow \mathsf{KeyGen}(params)$ and sets $status(u) = honest$ and $L(\breve{\mathsf{pk}}_0, u) = \infty$. $\mathcal{C}$ returns $\mathsf{pk}(u)$ to $\mathcal{A}$.

$\mathsf{SeeNym}(u)$: $\mathcal{A}$ invokes this oracle to see a fresh pseudonym for an honest node u. If $u \notin V(\breve{\mathsf{pk}}_0)$ or $status(u) \neq honest$, abort. Else, $\mathcal{C}$ runs NymGen to obtain $(\mathsf{nym}(u), \mathsf{aux}(u)) \leftarrow \mathsf{NymGen}(\mathsf{sk}(u), L(\breve{\mathsf{pk}}_0, u))$ and stores $\mathsf{nym}(u), \mathsf{aux}(u)$ at u. $\mathcal{C}$ returns $\mathsf{nym}(u)$ to $\mathcal{A}$.

$\mathsf{CertifyHonestParty}(\breve{\mathsf{pk}}_0, u, v)$: $\mathcal{A}$ invokes this oracle to have the honest party associated with u issue a credential to the honest party associated with v.

1. First, $\mathcal{C}$ checks that indeed u and v correspond to honest users and that v is ready to receive a credential from u, as follows:
   (a) If $u, v \notin V(\breve{\mathsf{pk}}_0)$, abort.
   (b) If $status(u) \neq honest$ or $status(v) \neq honest$, abort.
   (c) If $L(\breve{\mathsf{pk}}_0, v) \neq \infty$ (i.e. if v already has a credential), abort.
   (d) If $L(\breve{\mathsf{pk}}_0, u) = \infty$ (i.e. if u does not have a credential), abort.
2. Next, $\mathcal{A}$ selects a pseudonym $\mathsf{nym}(v)$ that he has seen for v under which v will interact with u.
3. If u is the root, then $\breve{\mathsf{pk}}_0$ is given as input to the Issue protocol instead of a pseudonym. More precisely, if $u = root$, then $\mathcal{C}$ runs:

$$[\mathsf{Issue}(0, \breve{\mathsf{pk}}_0, \breve{\mathsf{sk}}_0, \breve{\mathsf{pk}}_0, \perp, \perp, \mathsf{nym}(v)) \leftrightarrow$$

$$\mathsf{Receive}(0, \breve{\mathsf{pk}}_0, \mathsf{sk}(v), \mathsf{nym}(v), \mathsf{aux}(v), \breve{\mathsf{pk}}_0)] \to \mathsf{cred}_v$$

If u is not the root, $\mathcal{A}$ selects a pseudonym $\mathsf{nym}(u)$ that he has seen for u under which u will interact with v. Then, $\mathcal{C}$ runs:

$$[\mathsf{Issue}(L(\breve{\mathsf{pk}}_0, u), \breve{\mathsf{pk}}_0, \mathsf{sk}(u), \mathsf{nym}(u), \mathsf{aux}(u), \mathsf{cred}_u, \mathsf{nym}(v)) \leftrightarrow$$
$$\mathsf{Receive}(L(\breve{\mathsf{pk}}_0, u), \breve{\mathsf{pk}}_0, \mathsf{sk}(v), \mathsf{nym}(v), \mathsf{aux}(v), \mathsf{nym}(u))] \to \mathsf{cred}_v$$

4. $\mathcal{C}$ stores v's new credential cred_v at v, adds the edge (u, v) to $E(\breve{\mathsf{pk}}_0)$, and sets $L(\breve{\mathsf{pk}}_0, v) = L(\breve{\mathsf{pk}}_0, u) + 1$.

$\mathsf{VerifyCredFrom}(\breve{\mathsf{pk}}_0, u)$: The honest party associated with u proves to $\mathcal{A}$ that it has a credential at level $L(\breve{\mathsf{pk}}_0, u)$.

1. First, $\mathcal{C}$ checks that in fact u is honest and has a credential: if $u \notin V(\breve{\mathsf{pk}}_0)$, $\mathit{status}(u) \neq \mathit{honest}$, or $L(\breve{\mathsf{pk}}_0, u) = \infty$, abort.
2. Next, $\mathcal{A}$ selects a pseudonym $\mathsf{nym}(u)$ that he has seen for u.
3. Then, $\mathcal{C}$ runs the $\mathsf{CredProve}$ protocol with $\mathcal{A}$ as follows:

$$\mathsf{CredProve}(L(\breve{\mathsf{pk}}_0, u), \breve{\mathsf{pk}}_0, \mathsf{sk}(u), \mathsf{nym}(u), \mathsf{aux}(u), \mathsf{cred}_u) \leftrightarrow \mathcal{A}$$

$\mathsf{GetCredFrom}(\breve{\mathsf{pk}}_0, u, \mathsf{nym}_R)$: The honest party associated with u issues a credential to $\mathcal{A}$, whom it knows by nym_R.

1. First, $\mathcal{C}$ checks that in fact u is honest and has a credential: if $u \notin V(\breve{\mathsf{pk}}_0)$, $\mathit{status}(u) \neq \mathit{honest}$, or $L(\breve{\mathsf{pk}}_0, u) = \infty$, abort.
2. Next, $\mathcal{C}$ creates a new adversarial node v, sets $\mathit{status}(v) = \mathit{adversarial}$, and sets the identity to be $\hat{\mathsf{pk}}_v = f(\mathsf{nym}_R)$.
3. If u is the root, then $\mathcal{C}$ runs the Issue protocol with $\mathcal{A}$ as follows:

$$\mathsf{Issue}(0, \breve{\mathsf{pk}}_0, \breve{\mathsf{sk}}_0, \breve{\mathsf{pk}}_0, \bot, \bot, \mathsf{nym}_R) \leftrightarrow \mathcal{A}$$

If u is not the root, $\mathcal{A}$ selects a pseudonym $\mathsf{nym}(u)$ that he has seen for u, and $\mathcal{C}$ runs the Issue protocol with $\mathcal{A}$ as follows:

$$\mathsf{Issue}(L(\breve{\mathsf{pk}}_0, u), \breve{\mathsf{pk}}_0, \mathsf{sk}(u), \mathsf{nym}(u), \mathsf{aux}(u), \mathsf{cred}_u, \mathsf{nym}_R) \leftrightarrow \mathcal{A}$$

4. $\mathcal{C}$ adds the edge (u, v) to $E(\breve{\mathsf{pk}}_0)$ and sets $L(\breve{\mathsf{pk}}_0, v) = L(\breve{\mathsf{pk}}_0, u) + 1$.

$\mathsf{GiveCredTo}(\breve{\mathsf{pk}}_0, L_I(\breve{\mathsf{pk}}_0), \mathsf{nym}_I, v)$: $\mathcal{A}$ issues a credential to the honest party associated with v under a pseudonym nym_I (or $\breve{\mathsf{pk}}_0$ if he is the root).

1. If $v \notin V(\breve{\mathsf{pk}}_0)$, $\mathit{status}(v) \neq \mathit{honest}$, or $L(\breve{\mathsf{pk}}_0, v) \neq \infty$, abort.
2. $\mathcal{A}$ selects a pseudonym $\mathsf{nym}(v)$ that he has seen for v.
3. If $L_I(\breve{\mathsf{pk}}_0) = 0$, then $\mathcal{A}$ issues a credential to v acting as the root node (which he may do if $\mathit{status}(\mathit{root}) = \mathit{adversarial}$ in the game setup). $\mathcal{C}$ runs the $\mathsf{Receive}$ side of the protocol on behalf of the honest party associated with v. More precisely, cred_v is computed as follows:

$$[\mathcal{A} \leftrightarrow \mathsf{Receive}(0, \breve{\mathsf{pk}}_0, \mathsf{sk}(v), \mathsf{nym}(v), \mathsf{aux}(v), \breve{\mathsf{pk}}_0)] \to \mathsf{cred}_v$$

If $L_I(\breve{\mathsf{pk}}_0) \neq 0$, $\mathcal{A}$ issues a credential to v acting as some non-root node. $\mathcal{C}$ runs the $\mathsf{Receive}$ side of the protocol on behalf of the honest party associated with v, and cred_v is computed as follows:

$$[\mathcal{A} \leftrightarrow \mathsf{Receive}(L_I(\breve{\mathsf{pk}}_0), \breve{\mathsf{pk}}_0, \mathsf{sk}(v), \mathsf{nym}(v), \mathsf{aux}(v), \mathsf{nym}_I)] \rightarrow \mathsf{cred}_v$$

4. If $\mathsf{cred}_v \neq \perp$, $\mathcal{C}$ tells $\mathcal{A}$ that it didn't fail, stores v's new credential cred_v at v, and sets $L(\breve{\mathsf{pk}}_0, v) = L_I(\breve{\mathsf{pk}}_0) + 1$. Next, $\mathcal{C}$ computes the function f_{cred} on v's credential, $f_{cred}(\mathsf{cred}_v) = (\hat{\mathsf{pk}}_0, \hat{\mathsf{pk}}_1, \ldots, \hat{\mathsf{pk}}_{L_I})$, revealing the identities in v's credential chain. If according to $\mathcal{C}$'s data structure, there is some $\hat{\mathsf{pk}}_i$ in this chain such that $\hat{\mathsf{pk}}_i = f(\mathsf{nym}(u))$ for an honest user u, but $\hat{\mathsf{pk}}_{i+1} \neq f(\mathsf{nym}(v'))$ for any v' that received a credential from u, then $\mathcal{C}$ sets the `forgery` flag to true.
5. If $\mathsf{cred}_v \neq \perp$ and the `forgery` flag remains false, $\mathcal{C}$ fills in the gaps in the graph $G(\breve{\mathsf{pk}}_0)$ as follows. Starting from the nearest honest ancestor of v, $\mathcal{C}$ creates a new node for each (necessarily adversarial) identity in the chain between that honest node and v, sets $status = adversarial$ and the appropriate level for each node, and stores this information at each node along with its identity (e.g. $\hat{\mathsf{pk}}_j$). $\mathcal{C}$ then adds edges between the nodes on the chain from the nearest honest ancestor of v to v.

$\mathsf{DemoCred}(\breve{\mathsf{pk}}_0, L_P(\breve{\mathsf{pk}}_0), \mathsf{nym}_P)$: $\mathcal{A}$ proves possession of a credential at level $L_P(\breve{\mathsf{pk}}_0)$.

1. $\mathcal{C}$ runs the Verify side of the protocol with $\mathcal{A}$ as follows:

$$[\mathcal{A} \leftrightarrow \mathsf{CredVerify}(params, L_P(\breve{\mathsf{pk}}_0), \breve{\mathsf{pk}}_0, \mathsf{nym}_P)] \rightarrow \text{output (0 or 1)}$$

2. If output = 1, $\mathcal{C}$ tells $\mathcal{A}$ and computes the function f_{demo} on the transcript of the output, $f_{demo}(transcript) = (\hat{\mathsf{pk}}_0, \hat{\mathsf{pk}}_1, \ldots, \hat{\mathsf{pk}}_{L_P})$, revealing the identities in the credential chain. If according to $\mathcal{C}$'s data structure, there is some $\hat{\mathsf{pk}}_i$ in this chain such that $\hat{\mathsf{pk}}_i = f(\mathsf{nym}(u))$ for an honest user u, but $\hat{\mathsf{pk}}_{i+1} \neq f(\mathsf{nym}(v'))$ for any v' that received a credential from u, then $\mathcal{C}$ sets the `forgery` flag to true.
3. If output = 1 and the `forgery` flag remains false, $\mathcal{C}$ fills in the gaps in the graph $G(\breve{\mathsf{pk}}_0)$ as follows. $\mathcal{C}$ creates a new adversarial node v for the identity $\hat{\mathsf{pk}}_{L_P}$, sets $status(v) = adversarial$ and $L(\breve{\mathsf{pk}}_0, v) = L_P(\breve{\mathsf{pk}}_0)$, and stores this information at the node v along with its identity $\hat{\mathsf{pk}}_{L_P}$. Then, starting from the nearest honest ancestor of v, $\mathcal{C}$ creates a new node for each (necessarily adversarial) identity in the chain between that honest node and v, sets $status = adversarial$ and the appropriate level for each node, and stores this information at each node along with its identity (e.g. $\hat{\mathsf{pk}}_j$). $\mathcal{C}$ then adds edges between the nodes on the chain from the nearest honest ancestor of v to v.

$\mathsf{SetAnonChallenge}(u_0, u_1)$: If this oracle has ever been called before (so the status of the anonymity attack is not *undefined*), abort. Else, $\mathcal{A}$ will try to distinguish between the honest parties associated with u_0 and u_1. $\mathcal{C}$ checks that $u_0, u_1 \in V(\breve{\mathsf{pk}}_0)$ and $status(u_0) = status(u_1) = honest$. If these checks

pass, $\mathcal{C}$ sets the anonymity challenge pair to be (u_0, u_1) and updates the status of the anonymity attack to *defined*. Else, it updates the status of the anonymity attack to *forfeited*.

SeeNymAnon: $\mathcal{A}$ invokes this oracle to see fresh pseudonyms for u_b and $u_{\bar{b}}$. $\mathcal{C}$ runs NymGen to obtain $(\mathsf{nym}(u_b), \mathsf{aux}(u_b)) \leftarrow \mathsf{NymGen}(\mathsf{sk}(u_b), L(\breve{\mathsf{pk}}_0, u_b))$ and stores $\mathsf{nym}(u_b), \mathsf{aux}(u_b)$ at u_b. Similarly, $\mathcal{C}$ runs NymGen to obtain $(\mathsf{nym}(u_{\bar{b}}), \mathsf{aux}(u_{\bar{b}})) \leftarrow \mathsf{NymGen}(\mathsf{sk}(u_{\bar{b}}), L(\breve{\mathsf{pk}}_0, u_{\bar{b}}))$ and stores $\mathsf{nym}(u_{\bar{b}}), \mathsf{aux}(u_{\bar{b}})$ at $u_{\bar{b}}$. $\mathcal{C}$ adds the pair $(\mathsf{nym}(u_b), \mathsf{nym}(u_{\bar{b}}))$ to the set S and returns the pair to $\mathcal{A}$.

CertifyHonestAnon($\breve{\mathsf{pk}}_0, u$): $\mathcal{A}$ invokes this oracle to have the honest party associated with u issue credentials to u_b and $u_{\bar{b}}$.

1. If $u \notin V(\breve{\mathsf{pk}}_0)$, $\mathit{status}(u) \neq \mathit{honest}$, or $L(\breve{\mathsf{pk}}_0, u) = \infty$, abort. If $L(\breve{\mathsf{pk}}_0, u_b) = L(\breve{\mathsf{pk}}_0, u_{\bar{b}}) = \infty$ does not hold, abort (i.e. u_b and $u_{\bar{b}}$ must both be standalone nodes in $G(\breve{\mathsf{pk}}_0)$ so that they may receive credentials.)
2. $\mathcal{A}$ selects from the set S a pair of pseudonyms $(\mathsf{nym}(u_b), \mathsf{nym}(u_{\bar{b}}))$ that he has seen for u_b and $u_{\bar{b}}$.
3. If u is the root, then $\breve{\mathsf{pk}}_0$ is given as input to the Issue protocol instead of a pseudonym. More precisely, if $u = \mathit{root}$, then $\mathcal{C}$ runs:

$$[\mathsf{Issue}(0, \breve{\mathsf{pk}}_0, \breve{\mathsf{sk}}_0, \breve{\mathsf{pk}}_0, \bot, \bot, \mathsf{nym}(u_b)) \leftrightarrow \mathsf{Receive}(0, \breve{\mathsf{pk}}_0, \mathsf{sk}(u_b), \mathsf{nym}(u_b), \mathsf{aux}(u_b), \breve{\mathsf{pk}}_0)] \rightarrow \mathsf{cred}_{u_b}$$

   If u is not the root, $\mathcal{A}$ selects a pseudonym $\mathsf{nym}(u)$ that he has seen for u under which u will interact with u_b, and $\mathcal{C}$ runs:

$$[\mathsf{Issue}(L(\breve{\mathsf{pk}}_0, u), \breve{\mathsf{pk}}_0, \mathsf{sk}(u), \mathsf{nym}(u), \mathsf{aux}(u), \mathsf{cred}_u, \mathsf{nym}(u_b)) \leftrightarrow \mathsf{Receive}(L(\breve{\mathsf{pk}}_0, u), \breve{\mathsf{pk}}_0, \mathsf{sk}(u_b), \mathsf{nym}(u_b), \mathsf{aux}(u_b), \mathsf{nym}(u))] \rightarrow \mathsf{cred}_{u_b}$$

   $\mathcal{C}$ repeats this step for $u_{\bar{b}}$, using the same $\mathsf{nym}(u)$ (if $u \neq \mathit{root}$).
4. $\mathcal{C}$ stores u_b's new credential cred_{u_b} at u_b, adds the edge (u, u_b) to $E(\breve{\mathsf{pk}}_0)$, and sets $L(\breve{\mathsf{pk}}_0, u_b) = L(\breve{\mathsf{pk}}_0, u) + 1$. $\mathcal{C}$ repeats this step for $u_{\bar{b}}$.

CertifyAnonHonest($\breve{\mathsf{pk}}_0, b^*, v$): $\mathcal{A}$ invokes this oracle to have one of the anonymity challenge nodes, u_{b^*}, where $b^* = b$ or $\bar{b}$, issue a credential to the honest party associated with v.

1. If $v \notin V(\breve{\mathsf{pk}}_0)$, $\mathit{status}(v) \neq \mathit{honest}$, or $L(\breve{\mathsf{pk}}_0, v) \neq \infty$, abort. If $L(\breve{\mathsf{pk}}_0, u_{b^*}) = L(\breve{\mathsf{pk}}_0, u_{\bar{b}^*}) \neq \infty$ does not hold, abort (i.e. u_{b^*} and $u_{\bar{b}^*}$ must possess credentials at the same level under $\breve{\mathsf{pk}}_0$.)
2. $\mathcal{A}$ selects a pseudonym $\mathsf{nym}(v)$ that he has seen for v.
3. Note that u_{b^*} cannot be the root. $\mathcal{A}$ selects from the set S a pair of pseudonyms $(\mathsf{nym}(u_{b^*}), \mathsf{nym}(u_{\bar{b}^*}))$ that he has seen for u_{b^*} and $u_{\bar{b}^*}$. Then, $\mathcal{C}$ runs:

$$[\mathsf{Issue}(L(\breve{\mathsf{pk}}_0, u_{b^*}), \breve{\mathsf{pk}}_0, \mathsf{sk}(u_{b^*}), \mathsf{nym}(u_{b^*}), \mathsf{aux}(u_{b^*}), \mathsf{cred}_{u_{b^*}}, \mathsf{nym}(v)) \leftrightarrow \mathsf{Receive}(L(\breve{\mathsf{pk}}_0, u_{b^*}), \breve{\mathsf{pk}}_0, \mathsf{sk}(v), \mathsf{nym}(v), \mathsf{aux}(v), \mathsf{nym}(u_{b^*}))] \rightarrow \mathsf{cred}_v$$

4. $\mathcal{C}$ stores v's new credential cred_v at v, adds the edge (u_{b^*}, v) to $E(\breve{\mathsf{pk}}_0)$, and sets $L(\breve{\mathsf{pk}}_0, v) = L(\breve{\mathsf{pk}}_0, u_{b^*}) + 1$.

$\mathsf{VerifyCredFromAnon}(\breve{\mathsf{pk}}_0)$: The honest parties associated with u_b and $u_{\bar{b}}$ prove to $\mathcal{A}$ that they have credentials at level $L(\breve{\mathsf{pk}}_0, u_b) = L(\breve{\mathsf{pk}}_0, u_{\bar{b}})$.

1. If $L(\breve{\mathsf{pk}}_0, u_b) = L(\breve{\mathsf{pk}}_0, u_{\bar{b}}) \neq \infty$ does not hold, abort. Additionally, $\mathcal{C}$ checks that the two paths from u_b and $u_{\bar{b}}$ to the root $\breve{\mathsf{pk}}_0$ consist entirely of honest nodes, with the exception that $\breve{\mathsf{pk}}_0$ may be adversarial. If this check fails, $\mathcal{C}$ updates the status of the anonymity attack to *forfeited*.
2. Next, $\mathcal{A}$ selects from the set S a pair of pseudonyms $(\mathsf{nym}(u_b), \mathsf{nym}(u_{\bar{b}}))$ that he has seen for u_b and $u_{\bar{b}}$.
3. Then, $\mathcal{C}$ runs the $\mathsf{CredProve}$ protocol with $\mathcal{A}$ as follows:

$$\mathsf{CredProve}(L(\breve{\mathsf{pk}}_0, u_b), \breve{\mathsf{pk}}_0, \mathsf{sk}(u_b), \mathsf{nym}(u_b), \mathsf{aux}(u_b), \mathsf{cred}_{u_b}) \leftrightarrow \mathcal{A}$$

   $\mathcal{C}$ repeats this step for $u_{\bar{b}}$.

$\mathsf{GetCredFromAnon}(\breve{\mathsf{pk}}_0, b^*, \mathsf{nym}_R)$: The honest party associated with u_{b^*}, where $b^* = b$ or $\bar{b}$, issues a credential to $\mathcal{A}$, whom it knows by nym_R.

1. If $L(\breve{\mathsf{pk}}_0, u_{b^*}) = L(\breve{\mathsf{pk}}_0, u_{\bar{b}^*}) \neq \infty$ does not hold, abort. Additionally, $\mathcal{C}$ checks that the two paths from u_{b^*} and $u_{\bar{b}^*}$ to the root $\breve{\mathsf{pk}}_0$ consist entirely of honest nodes, with the exception that $\breve{\mathsf{pk}}_0$ may be adversarial. If this check fails, $\mathcal{C}$ updates the status of the anonymity attack to *forfeited*.
2. Next, $\mathcal{C}$ creates a new adversarial node v, sets $\mathit{status}(v) = \mathit{adversarial}$, and sets the identity to be $\hat{\mathsf{pk}}_v = f(\mathsf{nym}_R)$. Note that $\mathcal{A}$ can have $u_{b^*}, u_{\bar{b}^*}$ issue credentials to two different adversarial nodes, v, v', respectively, with the same underlying adversarial identity $\hat{\mathsf{pk}}_v = \hat{\mathsf{pk}}_{v'}$.
3. Note that u_{b^*} cannot be the root. $\mathcal{A}$ selects from the set S a pair of pseudonyms $(\mathsf{nym}(u_{b^*}), \mathsf{nym}(u_{\bar{b}^*}))$ that he has seen for u_{b^*} and $u_{\bar{b}^*}$. Then, $\mathcal{C}$ runs the Issue protocol with $\mathcal{A}$ as follows:

$$\mathsf{Issue}(L(\breve{\mathsf{pk}}_0, u_{b^*}), \breve{\mathsf{pk}}_0, \mathsf{sk}(u_{b^*}), \mathsf{nym}(u_{b^*}), \mathsf{aux}(u_{b^*}), \mathsf{cred}_{u_{b^*}}, \mathsf{nym}_R) \leftrightarrow \mathcal{A}$$

4. $\mathcal{C}$ adds the edge (u_{b^*}, v) to $E(\breve{\mathsf{pk}}_0)$ and sets $L(\breve{\mathsf{pk}}_0, v) = L(\breve{\mathsf{pk}}_0, u_{b^*}) + 1$.

$\mathsf{GiveCredToAnon}(\breve{\mathsf{pk}}_0, L_I(\breve{\mathsf{pk}}_0), \mathsf{nym}_I)$: $\mathcal{A}$ issues credentials to u_b and $u_{\bar{b}}$ under a pseudonym nym_I (or $\breve{\mathsf{pk}}_0$ if he is the root).

1. If $L(\breve{\mathsf{pk}}_0, u_b) = L(\breve{\mathsf{pk}}_0, u_{\bar{b}}) = \infty$ does not hold, abort.
2. $\mathcal{A}$ selects from the set S a pair of pseudonyms $(\mathsf{nym}(u_b), \mathsf{nym}(u_{\bar{b}}))$ that he has seen for u_b and $u_{\bar{b}}$.
3. If $L_I(\breve{\mathsf{pk}}_0) = 0$, then $\mathcal{A}$ issues a credential to u_b acting as the root node (which he may do if *root* = *adversarial* in the game setup). $\mathcal{C}$ runs the $\mathsf{Receive}$ side of the protocol on behalf of the honest party associated with u_b. More precisely, cred_{u_b} is computed as follows:

$$[\mathcal{A} \leftrightarrow \mathsf{Receive}(0, \breve{\mathsf{pk}}_0, \mathsf{sk}(u_b), \mathsf{nym}(u_b), \mathsf{aux}(u_b), \breve{\mathsf{pk}}_0)] \rightarrow \mathsf{cred}_{u_b}$$

If $L_I(\breve{\mathsf{pk}}_0) \neq 0$, $\mathcal{A}$ issues a credential to u_b acting as some non-root node. $\mathcal{C}$ runs the $\mathsf{Receive}$ side of the protocol on behalf of u_b, and cred_{u_b} is computed as follows:

$$[\mathcal{A} \leftrightarrow \mathsf{Receive}(L_I(\breve{\mathsf{pk}}_0), \breve{\mathsf{pk}}_0, \mathsf{sk}(u_b), \mathsf{nym}(u_b), \mathsf{aux}(u_b), \mathsf{nym}_I)] \rightarrow \mathsf{cred}_{u_b}$$

$\mathcal{C}$ repeats this step for $u_{\bar{b}}$, using the same nym_I (or $\breve{\mathsf{pk}}_0$).

4. If both $\mathsf{cred}_{u_b} \neq \perp$ and $\mathsf{cred}_{u_{\bar{b}}} \neq \perp$, $\mathcal{C}$ tells $\mathcal{A}$ that they didn't fail, stores u_b's new credential cred_{u_b} at u_b, and sets $L(\breve{\mathsf{pk}}_0, u_b) = L_I(\breve{\mathsf{pk}}_0) + 1$. Next, $\mathcal{C}$ computes the function f_{cred} on u_b's credential, $f_{cred}(\mathsf{cred}_{u_b}) = (\hat{\mathsf{pk}}_0, \hat{\mathsf{pk}}_1, \ldots, \hat{\mathsf{pk}}_{L_I})$, revealing the identities in u_b's credential chain. If according to $\mathcal{C}$'s data structure, there is some $\hat{\mathsf{pk}}_i$ in this chain such that $\hat{\mathsf{pk}}_i = f(\mathsf{nym}(u))$ for an honest user u, but $\hat{\mathsf{pk}}_{i+1} \neq f(\mathsf{nym}(v'))$ for any v' that received a credential from u, then $\mathcal{C}$ sets the `forgery` flag to true. $\mathcal{C}$ repeats this step for $u_{\bar{b}}$.
5. If both $\mathsf{cred}_{u_b} \neq \perp$ and $\mathsf{cred}_{u_{\bar{b}}} \neq \perp$ and the `forgery` flag remains false, $\mathcal{C}$ fills in the gaps in the graph $G(\breve{\mathsf{pk}}_0)$ as follows:
   (a) If there already exists an adversarial node v corresponding to the pseudonym nym_I with an edge connecting it to an honest parent, then $\mathcal{C}$ only adds an edge between v and u_b.
   (b) Else, starting from the nearest honest ancestor of u_b, $\mathcal{C}$ creates a new node for each (necessarily adversarial) identity in the chain between that honest node and u_b, sets *status* = *adversarial* and the appropriate level for each node, and stores this information at each node along with its identity (e.g. $\hat{\mathsf{pk}}_j$). $\mathcal{C}$ then adds edges between the nodes on the chain from the nearest honest ancestor of u_b to u_b. $\mathcal{C}$ repeats this step for $u_{\bar{b}}$.

$\mathsf{GuessAnon}(b')$: If this oracle has ever been called before or it is premature to call it (so the status of the anonymity attack is not *defined*), abort. If $b' = b$, the status of the anonymity attack is set to *success*. Else, the status is set to *fail*.

D Proof of Security for DAC

D.1 Unforgeability of DAC

Proof. (of Theorem 5.) We wish to show that if there exists a PPT adversary $\mathcal{A}$ such that there is non-negligible probability that the `forgery` flag will be set to true in the single-authority security game, then we can construct a PPT adversary $\mathcal{A}'$ that breaks unforgeability of mercurial signatures with non-negligible probability. There are two scenarios in which the `forgery` flag is set to true: (1) $\mathcal{A}$ proves possession of a credential at a level closer to the root than his own for a specific adversarial identity, or (2) $\mathcal{A}$ delegates a credential (possibly to a node in the anonymity challenge pair) at a level closer to the root than his own for a specific adversarial identity.

Supposing there exists such an adversary $\mathcal{A}$, we construct an adversary $\mathcal{A}'$ as a reduction $\mathcal{B}$ running $\mathcal{A}$ as a subroutine. We construct the reduction $\mathcal{B}$ for breaking unforgeability of mercurial signatures as follows.

$\mathcal{B}$ receives as input public parameters *params* and a fixed public key pk for one of the mercurial signature schemes, either MS_1 or MS_2, for which he will try to produce a forgery. He forwards *params* to $\mathcal{A}$, and the public key of the root, $\check{\mathsf{pk}}_0$, is established as in the single-authority security game by $\mathcal{B}$, who acts as $\mathcal{A}$'s challenger $\mathcal{C}$. As in the unforgeability game for mercurial signatures, $\mathcal{B}$ has access to a signing oracle $\mathsf{Sign}(\mathsf{sk}, \cdot)$, where sk is the secret key corresponding to pk.

$\mathcal{A}$ proceeds to make queries to the single-authority oracles. WLOG, let q_a be an upper bound, known a priori, on the number of queries $\mathcal{A}$ will make to the $\mathsf{AddHonestParty}$ oracle. Acting as the challenger for $\mathcal{A}$, $\mathcal{B}$ is responsible for computing responses to the oracle queries, forwarding them to $\mathcal{A}$, and updating the graph $G(\check{\mathsf{pk}}_0)$ with the appropriate information. $\mathcal{B}$ maintains the graph as $\mathcal{A}$'s challenger $\mathcal{C}$ would, but without any edges. $\mathcal{B}$ does not have oracle access to the (hard-to-compute) functions f, f_{cred}, and f_{demo} required to maintain edges correctly.

For one of $\mathcal{A}$'s $\mathsf{AddHonestParty}$ queries, chosen uniformly at random from among all q_a such queries, $\mathcal{B}$ makes his challenge pk the public key for that user u^* and updates the graph with the new node. For a query to the SeeNym oracle on u^*, $\mathcal{B}$ forms a new pseudonym nym from pk by running $\mathsf{ConvertPK}$.

Suppose $\mathcal{A}$ subsequently makes a query to either the $\mathsf{CertifyHonestParty}$ oracle or the $\mathsf{GiveCredTo}$ oracle to have u^* receive a credential. If u^* is at an even level and pk is odd, or vice versa, abort. This will lead the reduction to abort, independently of the adversary's view, with probability $1/2$. If $\mathcal{B}$ has not aborted, it proceeds as follows. In the case of $\mathsf{CertifyHonestParty}$, $\mathcal{B}$ has all the information required to run exactly the same steps as the challenger $\mathcal{C}$ would. In the case of $\mathsf{GiveCredTo}$, $\mathcal{B}$ forms a new pseudonym nym from pk by running $\mathsf{ConvertPK}$ and invokes the zero-knowledge simulator to prove to $\mathcal{A}$ that he knows the secret key associated with nym. Then, $\mathcal{B}$ receives from $\mathcal{A}$ a certification chain for nym.

If $\mathcal{A}$ subsequently queries the $\mathsf{CertifyHonestParty}$ oracle or the $\mathsf{GetCredFrom}$ oracle to have u^* delegate a credential, $\mathcal{B}$ must forward that request to his signing oracle since he does not know the corresponding secret key sk.

If $\mathcal{A}$ queries $\mathsf{VerifyCredFrom}(u^*)$, $\mathcal{B}$ randomizes u^*'s certification chain correctly and invokes the zero-knowledge simulator to convince $\mathcal{A}$ that he knows u^*'s secret key.

For $\mathcal{A}$'s oracle queries that do not take u^* as input, $\mathcal{B}$ has all the information required to run exactly the same steps as the challenger $\mathcal{C}$ would, with the exception that $\mathcal{B}$ does not have oracle access to the (hard-to-compute) functions f, f_{cred}, and f_{demo}.

Suppose $\mathcal{A}$ forges at least one credential chain, and consider $\mathcal{A}$'s first forgery. Either when showing or when issuing a credential (possibly to a node in the anonymity challenge pair), $\mathcal{A}$ produces a certification chain $(\mathsf{nym}_0, \mathsf{nym}_1, \ldots, \mathsf{nym}_n)$ such that: (1) the underlying identities of $(\mathsf{nym}_0, \mathsf{nym}_1, \ldots, \mathsf{nym}_n)$ are

$(\hat{\mathsf{pk}}_0, \hat{\mathsf{pk}}_1, \ldots, \hat{\mathsf{pk}}_A)$, ending in $\mathcal{A}$'s identity $\hat{\mathsf{pk}}_A$, (2) $\hat{\mathsf{pk}}_0 = \breve{\mathsf{pk}}_0$, and (3) there is some $\hat{\mathsf{pk}}_i$ in this chain such that $\hat{\mathsf{pk}}_i = f(\mathsf{nym}(u))$ for an honest user u, but $\hat{\mathsf{pk}}_{i+1} \neq f(\mathsf{nym}(v'))$ for any v' that received a credential from u.

The `forgery` flag must be set to `true` the first time $\mathcal{A}$ forges a credential chain or $\mathcal{A}$ will know he is operating inside the reduction, and the reduction will fail. Up until $\mathcal{A}$ forges a credential chain, the view that $\mathcal{A}$ receives in his interaction with $\mathcal{B}$ is exactly the same view that he would receive in the security game. Indeed, the description of $\mathcal{B}$'s responses to $\mathcal{A}$'s oracle queries above demonstrates that $\mathcal{B}$ is able to compute each step of each of $\mathcal{A}$'s oracle queries exactly as $\mathcal{A}$'s challenger in the security game would, with one exception: $\mathcal{B}$ does not have oracle access to the (hard-to-compute) functions f, f_{cred}, and f_{demo}. However, these are internal, bookkeeping functions that are only needed to establish whether or not $\mathcal{A}$ has successfully forged a credential. Thus, even if $\mathcal{B}$ does not carry them out correctly, it's hidden from $\mathcal{A}$ as long as $\mathcal{A}$'s first forged credential chain is caught.

Using $\mathcal{A}$'s forged credential chain, $\mathcal{B}$ constructs his forgery of a mercurial signature as follows. With probability $1/2$, $\mathcal{B}$ picks i uniformly at random from $\{0, 1, \ldots, n-1\}$ and outputs his forgery as $(\mathsf{pk}^*, M^*, \sigma^*) = (\mathsf{nym}_i, \mathsf{nym}_{i+1}, \sigma_{i+1})$, where $\mathsf{nym}_0 = \breve{\mathsf{pk}}_0$. This is the "forged chain" case.

With remaining probability $1/2$: (1) if the forgery occurred as part of $\mathsf{GiveCredTo}$, $\mathcal{B}$ outputs $(\mathsf{pk}^*, M^*, \sigma^*) = (\mathsf{nym}_n, \mathsf{nym}_R, \sigma_R)$, where nym_R is the recipient's pseudonym and σ_R is the signature $\mathcal{A}$ issued to nym_R under his nym_n; (2) if the forgery occurred as part of $\mathsf{DemoCred}$, $\mathcal{B}$ uses the knowledge extractor to extract sk_n that corresponds to nym_n, computes a pk^* that corresponds to sk_n, and uses the corresponding key pair to sign a random message. He outputs the resulting $(\mathsf{pk}^*, M^*, \sigma^*)$. This is the "forged identity" case.

Now, let us analyze the likelihood that $\mathcal{B}$'s forgery is successful. Note that, given that $\mathcal{B}$ has not aborted, the identity of u^* is independent of $\mathcal{A}$'s view.

By inspecting $(\mathsf{nym}_0, \mathsf{nym}_1, \ldots, \mathsf{nym}_n)$, we could relate the pseudonyms to the underlying public keys, if they exist, of the honest parties under $\mathcal{B}$'s control (this cannot be done efficiently, but we don't have to worry about efficiency here.) If each of these public keys exists and is under $\mathcal{B}$'s control, then nym_n is equivalent to the public key of some honest user u. With probability $1/q_a$, this user is u^*. Since $\mathcal{A}$ succeeded in either proving knowledge of the secret key corresponding to nym_n or issuing a new signature on behalf of nym_n, if $\mathcal{B}$ acts as in the "forged identity" case (which he does with probability $1/2$), he successfully breaks unforgeability of the mercurial signature scheme.

Suppose that in fact one of the pseudonyms on the list $(\mathsf{nym}_0, \mathsf{nym}_1, \ldots, \mathsf{nym}_n)$ does not correspond to an honest identity. Suppose the first one in the list that doesn't correspond to an honest user's identity is nym_{i+1}. Then $(\mathsf{nym}_i, \mathsf{nym}_{i+1}, \sigma_{i+1})$ is a successful forgery for the mercurial signature as long as $\mathsf{nym}_i \in [\mathsf{pk}]_{\mathcal{R}_{\mathsf{pk}}}$, which is true with probability $1/q_a$. Thus, if $\mathcal{B}$ acts as in the "forged chain" case (which he does with probability $1/2$), he successfully breaks unforgeability of the mercurial signature scheme.

Putting everything together, if $\mathcal{A}$'s success probability is ϵ, $\mathcal{B}$'s success probability is $\epsilon/4q_a$. If the mercurial signature scheme is unforgeable, $\mathcal{A}$'s success

probability must be negligible. Therefore, the DAC construction is unforgeable in the single-authority setting.

D.2 Anonymity of DAC

Proof. (of Theorem 5, continued.) Let $\Gamma(k)$ be a polynomial. For $0 \leq i \leq \Gamma(k)$, let $\mathcal{H}_i$ be the hybrid experiment defined as the following modified single-authority security game.

For an adversary $\mathcal{A}$'s j^{th} query to $\mathsf{AddHonestParty}$, $j \leq i$, the challenger creates a new, honest node with some fixed underlying odd public key $\mathsf{pk}_{\mathsf{odd}}$, which he randomizes using $\mathsf{ConvertPK}$, and a distinct even public key. For the j^{th} query to $\mathsf{AddHonestParty}$, $j > i$, the oracle creates a new, honest node with distinct even and odd public keys. All of the other single-authority oracles are the same as in the single-authority security game.

By definition, $\mathcal{H}_0$ (and $\mathcal{H}_1$) corresponds to the security game in which all honest nodes have distinct identities (i.e. the real single-authority security game), while $\mathcal{H}_{\Gamma(k)}$ corresponds to the security game in which all odd-level honest nodes have the same underlying identity $\mathsf{pk}_{\mathsf{odd}}$ and all even-level nodes have distinct identities. By the hybrid argument, it is sufficient to show that for every polynomial Γ and for every probabilistic, polynomial-time adversary making at most $\Gamma(k)$ queries to the $\mathsf{AddHonestParty}$ oracle, there exists a negligible function $\nu(k)$ such that for every k and for every i such that $0 \leq i \leq \Gamma(k) - 1$, the advantage of the adversary in distinguishing $\mathcal{H}_i$ from $\mathcal{H}_{i+1}$ is at most $\nu(k)$.

Suppose that for the functions f, f_{cred}, and f_{demo} (not necessarily easy to compute), there exists a polynomial Γ and a PPT adversary $\mathcal{A}$ making at most $\Gamma(k)$ queries to the $\mathsf{AddHonestParty}$ oracle in the single-authority hybrid security game such that there exists a non-negligible $\epsilon(k)$ such that for some $i(k)$, $\mathcal{A}$ can distinguish $\mathcal{H}_{i(k)}$ from $\mathcal{H}_{i(k)+1}$ with advantage $\epsilon(k)$ for the functions f, f_{cred}, and f_{demo}. Then, let us show that there exists a probabilistic, polynomial-time $\mathcal{B}$ that breaks public key class-hiding of mercurial signatures. We construct $\mathcal{B}$ as a reduction running $\mathcal{A}$ as a subroutine. $\mathcal{B}$ serves as the challenger for $\mathcal{A}$ in the single-authority hybrid security game and as the adversary for his own challenger in the public key class-hiding game for mercurial signatures.

We construct the reduction $\mathcal{B}$ for breaking public key class-hiding of mercurial signatures as follows. $\mathcal{B}$ receives as input public parameters *params* and two fixed public keys $\mathsf{pk}_1, \mathsf{pk}_2$. $\mathcal{B}$ will try to distinguish whether or not they belong to the same equivalence class. He forwards *params* to $\mathcal{A}$, and the public key of the root, $\breve{\mathsf{pk}}_0$, is established in the usual way by $\mathcal{B}$, who acts as $\mathcal{A}$'s challenger $\mathcal{C}$. As in the public key class-hiding game for mercurial signatures, $\mathcal{B}$ has access to signing oracles $\mathsf{Sign}(\mathsf{sk}_1, \cdot), \mathsf{Sign}(\mathsf{sk}_2, \cdot)$, where $\mathsf{sk}_1, \mathsf{sk}_2$ are the secret keys corresponding to $\mathsf{pk}_1, \mathsf{pk}_2$, respectively.

$\mathcal{A}$ proceeds to make queries to the single-authority oracles. Acting as the challenger for $\mathcal{A}$, $\mathcal{B}$ is responsible for computing the responses to the oracle queries, forwarding them to $\mathcal{A}$, and updating the graph $G(\breve{\mathsf{pk}}_0)$ with the appropriate information. $\mathcal{B}$ maintains the graph as $\mathcal{A}$'s challenger $\mathcal{C}$ would, but without any

edges. $\mathcal{B}$ does not have oracle access to the (hard-to-compute) functions f, f_{cred}, and f_{demo} required to maintain edges correctly.

$\mathcal{B}$ responds to the single-authority oracle queries as follows:

$\mathsf{AddHonestParty}(u)$: The adversary $\mathcal{A}$ invokes this oracle to create a new, honest node u. Let the current graph be $G(\breve{\mathsf{pk}}_0) = (V(\breve{\mathsf{pk}}_0), E(\breve{\mathsf{pk}}_0))$. If $u \in V(G(\breve{\mathsf{pk}}_0))$ (i.e., it is not a new node), abort. Else, $\mathcal{B}$ adds a new node u to $G(\breve{\mathsf{pk}}_0)$ (so $G(\breve{\mathsf{pk}}_0) := (V(\breve{\mathsf{pk}}_0) \cup \{u\}, E(\breve{\mathsf{pk}}_0))$) for the j^{th} query as follows.

(1) $j \leq i$: $\mathcal{B}$ samples a key converter $\rho \leftarrow \mathsf{sample}_\rho$ and runs $\mathsf{ConvertPK}$ to obtain $\tilde{\mathsf{pk}}_1 \leftarrow \mathsf{ConvertPK}(\mathsf{pk}_1, \rho)$. $\mathcal{B}$ sets $status(u) = honest$, $\mathsf{pk}(u) = \tilde{\mathsf{pk}}_1$, $\mathsf{aux}(u) = \rho$, and $L(\breve{\mathsf{pk}}_0, u) = \infty$. $\mathcal{B}$ returns $\mathsf{pk}(u)$ to $\mathcal{A}$.
(2) $j = i + 1$: $\mathcal{B}$ sets $status(u) = honest$, $\mathsf{pk}(u) = \mathsf{pk}_2$, and $L(\breve{\mathsf{pk}}_0, u) = \infty$. $\mathcal{B}$ returns $\mathsf{pk}(u)$ to $\mathcal{A}$.
(3) $j > i + 1$: $\mathcal{B}$ runs KeyGen to obtain $(\mathsf{pk}(u), \mathsf{sk}(u)) \leftarrow \mathsf{KeyGen}(1^k)$ and sets $status(u) = honest$ and $L(\breve{\mathsf{pk}}_0, u) = \infty$. $\mathcal{B}$ returns $\mathsf{pk}(u)$ to $\mathcal{A}$.

$\mathsf{SeeNym}(u)$: The adversary $\mathcal{A}$ invokes this oracle to see a fresh pseudonym for an honest node u. If $u \notin V(G(\breve{\mathsf{pk}}_0))$ or $status(u) \neq honest$, abort. Else, $\mathcal{B}$ computes a pseudonym for u as follows. First, $\mathcal{B}$ samples a key converter $\rho \leftarrow \mathsf{sample}_\rho$. $\mathcal{B}$ runs $\mathsf{ConvertPK}$ to obtain $\tilde{\mathsf{pk}} \leftarrow \mathsf{ConvertPK}(\mathsf{pk}(u), \rho)$, where $\mathsf{pk}(u)$ has underlying public key $\mathsf{pk}_1, \mathsf{pk}_2$, or some distinct public key. $\mathcal{B}$ sets $\mathsf{nym}(u) = \tilde{\mathsf{pk}}$, $\mathsf{aux}(u) = \rho$ and stores $\mathsf{nym}(u), \mathsf{aux}(u)$ at u. $\mathcal{B}$ returns $\mathsf{nym}(u)$ to $\mathcal{A}$.

$\mathsf{CertifyHonestParty}(\breve{\mathsf{pk}}_0, u, v)$: The adversary $\mathcal{A}$ invokes this oracle to have the honest party associated with u issue a credential to the honest party associated with v.

First, $\mathcal{A}$ selects two nodes, u and v, and $\mathcal{B}$ carries out the required checks in step 1, which he can do because for each node, he maintains its status, *honest* or *adversarial*, and its level in $G(\breve{\mathsf{pk}}_0)$. In step 2, $\mathcal{A}$ selects pseudonyms $\mathsf{nym}(u), \mathsf{nym}(v)$ under which u and v will interact with one another (unless u is the root). In step 3, $\mathcal{B}$ is responsible for computing:

$$[\mathsf{Issue}(L(\breve{\mathsf{pk}}_0, u), \breve{\mathsf{pk}}_0, \mathsf{sk}(u), \mathsf{nym}(u), \mathsf{aux}(u), \mathsf{cred}_u, \mathsf{nym}(v)) \leftrightarrow \mathsf{Receive}(L(\breve{\mathsf{pk}}_0, u), \breve{\mathsf{pk}}_0, \mathsf{sk}(v), \mathsf{nym}(v), \mathsf{aux}(v), \mathsf{nym}(u))] \rightarrow \mathsf{cred}_v$$

$\mathcal{B}$ runs both the Issue and the $\mathsf{Receive}$ side, producing cred_v, as follows.
First, $\mathcal{B}$ computes u's randomized credential chain: $\{(\mathsf{nym}'_1, \ldots, \mathsf{nym}(u)'), (\sigma'_1, \ldots, \sigma'_{L(u)})\}$.
$\mathcal{B}$ then computes $\sigma_{L(u)+1} = \mathsf{Sign}(\mathsf{sk}(u), \mathsf{nym}(v))$ as follows. If u's underlying public key is pk_1 or pk_2, $\mathcal{B}$ invokes the appropriate mercurial signing oracle, $\mathsf{Sign}(\mathsf{sk}_1, \mathsf{nym}(v))$ or $\mathsf{Sign}(\mathsf{sk}_2, \mathsf{nym}(v))$, respectively; otherwise, $\mathcal{B}$ computes $\mathsf{Sign}(\mathsf{sk}(u), \mathsf{nym}(v))$ himself.
The resulting credential chain is $\mathsf{cred}_v : \{(\mathsf{nym}'_1, \ldots, \mathsf{nym}(u)', \mathsf{nym}(v)), (\sigma'_1, \ldots, \sigma'_{L(u)}, \sigma_{L(u)+1})\}$.

In step 4, $\mathcal{B}$ stores v's new credential cred_v at v and sets $L(\breve{\mathsf{pk}}_0, v) = L(\breve{\mathsf{pk}}_0, u) + 1$.

$\mathsf{VerifyCredFrom}(\breve{\mathsf{pk}}_0, u)$: The honest party associated with u proves to $\mathcal{A}$ that it has a credential at level $L(\breve{\mathsf{pk}}_0, u)$.

First, $\mathcal{A}$ selects a node u, and $\mathcal{B}$ carries out the required checks in step 1, which he can do because for each node, he maintains its status, *honest* or *adversarial*, and its level in $G(\breve{\mathsf{pk}}_0)$. In step 2, $\mathcal{A}$ selects a pseudonym $\mathsf{nym}(u)$ under which u will prove posession of its credential. In step 3, $\mathcal{B}$ is responsible for computing:

$$\mathsf{CredProve}(L(\breve{\mathsf{pk}}_0, u), \breve{\mathsf{pk}}_0, \mathsf{sk}(u), \mathsf{nym}(u), \mathsf{aux}(u), \mathsf{cred}_u) \leftrightarrow \mathcal{A}$$

$\mathcal{B}$ runs the $\mathsf{CredProve}$ side as follows.

$\mathcal{B}$ randomizes u's credential chain $\mathsf{cred}_u : \{(\mathsf{nym}_1, \ldots, \mathsf{nym}(u)), (\sigma_1, \ldots, \sigma_{L(u)})\}$ and sends $\{(\mathsf{nym}'_1, \ldots, \mathsf{nym}(u)'), (\sigma'_1, \ldots, \sigma'_{L(u)})\}$ to $\mathcal{A}$. $\mathcal{B}$ then proves knowledge of the secret key that corresponds to the last pseudonym in the chain, $\mathsf{nym}(u)'$, as follows. If u's public key is pk_1 or pk_2, $\mathcal{B}$ invokes the zero-knowledge simulator to convince $\mathcal{A}$ that he knows u's secret key. Otherwise, he proves knowledge of $\mathsf{sk}(u)$ directly.

$\mathsf{GetCredFrom}(\breve{\mathsf{pk}}_0, u, \mathsf{nym}_R)$: The honest party associated with u issues a credential to the adversary, whom it knows by nym_R.

First, $\mathcal{A}$ selects a node u and provides an adversarial pseudonym nym_R. Then, $\mathcal{B}$ carries out the required checks in step 1. $\mathcal{B}$ is not able to carry out step 2 as written, as he does not know the (hard-to-compute) function f, but he does create an adversarial node v associated with nym_R.

In step 3, $\mathcal{B}$ is responsible for computing:

$$\mathsf{Issue}(L(\breve{\mathsf{pk}}_0, u), \breve{\mathsf{pk}}_0, \mathsf{sk}(u), \mathsf{nym}(u), \mathsf{aux}(u), \mathsf{cred}_u, \mathsf{nym}_R) \leftrightarrow \mathcal{A}$$

$\mathcal{B}$ runs the Issue side as follows.

$\mathcal{B}$ randomizes u's credential chain cred_u and computes $\sigma_{L(u)+1}$ the exact same way as in $\mathsf{CertifyHonestParty}$. $\mathcal{B}$ sends $\mathsf{cred}_v = \{(\mathsf{nym}'_1, \ldots, \mathsf{nym}(u)', \mathsf{nym}_R), (\sigma'_1, \ldots, \sigma'_{L(u)}, \sigma_{L(u)+1})\}$ to $\mathcal{A}$.

In step 4, $\mathcal{B}$ sets $L(\breve{\mathsf{pk}}_0, v) = L(\breve{\mathsf{pk}}_0, u) + 1$.

$\mathsf{GiveCredTo}(\breve{\mathsf{pk}}_0, L_I(\breve{\mathsf{pk}}_0), \mathsf{nym}_I, v)$: $\mathcal{A}$ issues a credential to the honest party associated with v under a pseudonym nym_I (or $\breve{\mathsf{pk}}_0$ if he is the root).

First, $\mathcal{A}$ selects a node v and provides an adversarial pseudonym nym_I and level $L_I(\breve{\mathsf{pk}}_0)$. Then, $\mathcal{B}$ carries out the required checks in step 1. In step 2, $\mathcal{A}$ selects a pseudonym $\mathsf{nym}(v)$ under which v will interact with the adversary. In step 3, $\mathcal{B}$ is responsible for computing:

$$[\mathcal{A} \leftrightarrow \mathsf{Receive}(L_I(\breve{\mathsf{pk}}_0), \breve{\mathsf{pk}}_0, \mathsf{sk}(v), \mathsf{nym}(v), \mathsf{aux}(v), \mathsf{nym}_I)] \rightarrow \mathsf{cred}_v$$

$\mathcal{B}$ runs the $\mathsf{Receive}$ side as follows.

First, $\mathcal{B}$ proves to $\mathcal{A}$ that he knows the secret key $\mathsf{sk}(v)$ corresponding to $\mathsf{nym}(v)$ as follows. If v's public key is pk_1 or pk_2, $\mathcal{B}$ invokes the zero-knowledge

simulator to convince $\mathcal{A}$ that he knows v's secret key. Otherwise, he proves knowledge of $\mathsf{sk}(v)$ directly. He then receives from $\mathcal{A}$ a credential chain cred_v. In step 4, if $\mathsf{cred}_v \neq \perp$, $\mathcal{B}$ tells $\mathcal{A}$ it didn't fail, stores v's new credential cred_v at v, and sets $L(\breve{\mathsf{pk}}_0, v) = L_I(\breve{\mathsf{pk}}_0) + 1$. $\mathcal{B}$ cannot compute the (hard-to-compute) function f_{cred} on v's credential to reveal the identities on v's credential chain and therefore cannot fill in the gaps in $G(\breve{\mathsf{pk}}_0)$ as in step 5.

$\mathsf{DemoCred}(\breve{\mathsf{pk}}_0, L_P(\breve{\mathsf{pk}}_0), \mathsf{nym}_P)$: $\mathcal{A}$ proves possession of a credential at level $L_P(\breve{\mathsf{pk}}_0)$.

First, $\mathcal{A}$ provides an adversarial pseudonym nym_P and level $L_P(\breve{\mathsf{pk}}_0)$. In step 1, $\mathcal{B}$ is responsible for computing:

$$[\mathcal{A} \leftrightarrow \mathsf{CredVerify}(\mathit{params}, L_P(\breve{\mathsf{pk}}_0), \breve{\mathsf{pk}}_0, \mathsf{nym}_P)] \rightarrow \text{output (0 or 1)}$$

$\mathcal{B}$ knows $\mathit{params}, L_P(\breve{\mathsf{pk}}_0), \breve{\mathsf{pk}}_0$, and nym_P, so he has all of the information required to run the $\mathsf{CredVerify}$ protocol. If $\mathsf{cred}_P \neq \perp$, $\mathcal{B}$ outputs 1 and tells $\mathcal{A}$; otherwise, he outputs 0. In step 2, $\mathcal{B}$ cannot compute the (hard-to-compute) function f_{demo} on the transcript of the output to reveal the identities on the credential chain and therefore cannot fill in the gaps in $G(\breve{\mathsf{pk}}_0)$ as in step 3. However, he creates a new, adversarial node v corresponding to nym_P and sets $L(\breve{\mathsf{pk}}_0, v) = L_P(\breve{\mathsf{pk}}_0)$.

The remaining oracles in the security game are invoked during the anonymity challenge. $\mathcal{B}$ handles the anonymity challenge by selecting a random value for an "anonymity bit" β, $\beta \leftarrow \{0, 1\}$. $\mathcal{B}$ responds to each anonymity oracle query as follows:

$\mathsf{SetAnonChallenge}(u_0, u_1)$: $\mathcal{B}$ sets the anonymity challenge nodes u_0, u_1. That is, $\mathcal{A}$ selects two nodes, u_0 and u_1, and $\mathcal{B}$ carries out the necessary checks, which he can do because for each node, he maintains its status, *honest* or *adversarial*,.

$\mathsf{SeeNymAnon}$: $\mathcal{B}$ executes $\mathsf{SeeNym}(u_\beta)$ and $\mathsf{SeeNym}(u_{\bar{\beta}})$, where $\bar{\beta} = 1 - \beta$.

$\mathsf{CertifyHonestAnon}(\breve{\mathsf{pk}}_0, u)$: $\mathcal{B}$ executes $\mathsf{CertifyHonestParty}(\breve{\mathsf{pk}}_0, u, u_\beta)$ and $\mathsf{CertifyHonestParty}(\breve{\mathsf{pk}}_0, u, u_{\bar{\beta}})$.

$\mathsf{CertifyAnonHonest}(\breve{\mathsf{pk}}_0, b^*, v)$: If $b^* = \beta$, $\mathcal{B}$ executes $\mathsf{CertifyHonestParty}(\breve{\mathsf{pk}}_0, u_\beta, v)$. If $b^* = \bar{\beta}$, $\mathcal{B}$ executes $\mathsf{CertifyHonestParty}(\breve{\mathsf{pk}}_0, u_{\bar{\beta}}, v)$.

$\mathsf{VerifyCredFromAnon}(\breve{\mathsf{pk}}_0)$: $\mathcal{B}$ executes $\mathsf{VerifyCredFrom}(\breve{\mathsf{pk}}_0, u_\beta)$ and $\mathsf{VerifyCredFrom}(\breve{\mathsf{pk}}_0, u_{\bar{\beta}})$.

$\mathsf{GetCredFromAnon}(\breve{\mathsf{pk}}_0, b^*, \mathsf{nym}_R)$: If $b^* = \beta$, $\mathcal{B}$ executes $\mathsf{GetCredFrom}(\breve{\mathsf{pk}}_0, u_\beta, \mathsf{nym}_R)$. If $b^* = \bar{\beta}$, $\mathcal{B}$ executes $\mathsf{GetCredFrom}(\breve{\mathsf{pk}}_0, u_{\bar{\beta}}, \mathsf{nym}_R)$.

$\mathsf{GiveCredToAnon}(\breve{\mathsf{pk}}_0, L_I(\breve{\mathsf{pk}}_0), \mathsf{nym}_I)$: $\mathcal{B}$ executes $\mathsf{GiveCredTo}(\breve{\mathsf{pk}}_0, L_I(\breve{\mathsf{pk}}_0), \mathsf{nym}_I, u_\beta)$ and $\mathsf{GiveCredTo}(\breve{\mathsf{pk}}_0, L_I(\breve{\mathsf{pk}}_0), \mathsf{nym}_I, u_{\bar{\beta}})$. Note that the `forgery` flag may be set to `true` during these executions.

$\mathsf{GuessAnon}(b')$: If this oracle has ever been called before, or it is premature to call it (so the status of the anonymity attack is not *defined*), abort. If $b' = \beta$, $\mathcal{B}$ sets the status of the anonymity attack is set to *success*. Else, $\mathcal{B}$ sets the status to *fail*.

The above description of $\mathcal{B}$'s responses to $\mathcal{A}$'s oracle queries demonstrates that $\mathcal{B}$ is able to emulate the appropriate hybrid and compute each step of each of $\mathcal{A}$'s oracle queries exactly as $\mathcal{A}$'s challenger in the security game would, with one exception: $\mathcal{B}$ does not have oracle access to the (hard-to-compute) functions f, f_{cred}, and f_{demo}. Also, note that in the description of $\mathcal{B}$'s responses, we have substituted the zero-knowledge prover with a zero-knowledge simulator, which is standard.

Claim 6 *For all $0 \leq i \leq \Gamma(k) - 1$, $\mathcal{A}$ cannot distinguish $\mathcal{H}_i$ from $\mathcal{H}_{i+1}$.*

There exists a node u with underlying public key pk_2 in hybrid $\mathcal{H}_i$ but with underlying public key pk_1 in hybrid $\mathcal{H}_{i+1}$, by definition. Thus, $\mathcal{A}$'s non-negligible probability of distinguishing the hybrids translates into $\mathcal{B}$'s success probability of determining if pk_1 and pk_2 are in the same equivalence class in the public key class-hiding game. We have therefore constructed a PPT algorithm $\mathcal{B}$ that breaks the public key class-hiding game for the functions f, f_{cred}, and f_{demo} in the single-authority setting.

Since $\mathcal{B}$ does not have oracle access to the (hard-to-compute) functions f, f_{cred}, and f_{demo}, he is unable to catch forgeries. However, $\mathcal{A}$ cannot forge in hybrid $\mathcal{H}_0$ because it corresponds to the security game in which all honest nodes have distinct identities (i.e. the real single-authority security game). Since all hybrids are indistinguishable by Claim 6, forgeries do not occur in any hybrid.

Now, let $\tilde{\mathcal{H}}_i$ be the hybrid experiment defined as the following modified single-authority security game.

For an adversary $\mathcal{A}$'s j^{th} query to $\mathsf{AddHonestParty}$, $j \leq i$, the challenger creates a new, honest node with some fixed underlying odd public key $\mathsf{pk}_{\mathsf{odd}}$, and a some fixed underlying even public key $\mathsf{pk}_{\mathsf{even}}$, both of which he randomizes using $\mathsf{ConvertPK}$. For the j^{th} query to $\mathsf{AddHonestParty}$, $j > i$, the oracle creates a new, honest node with distinct even and odd public keys. All of the other single-authority oracles are the same as in the single-authority security game.

By definition, $\tilde{\mathcal{H}}_0$ (and $\tilde{\mathcal{H}}_1$) corresponds to the security game in which all odd-level honest nodes have the same underlying identity $\mathsf{pk}_{\mathsf{odd}}$ and all even-level nodes have distinct identities (i.e. hybrid $\mathcal{H}_{\Gamma(k)}$ from before), while $\mathcal{H}_{\Gamma(k)}$ corresponds to the security game in which all odd-level honest nodes have the same underlying identity $\mathsf{pk}_{\mathsf{odd}}$ and all even-level nodes have the same underlying $\mathsf{pk}_{\mathsf{even}}$.

We construct a reduction $\mathcal{B}$ for breaking public key class-hiding of mercurial signatures as before, where $\mathcal{B}$ answers the single-authority oracle queries with the appropriate underlying public keys as described above. $\mathcal{A}$ cannot distinguish hybrids or else $\mathcal{B}$ can break public key class-hiding.

Claim 7 *$\mathcal{A}$'s view is independent on the anonymity bit.*

$\mathcal{H}_{\Delta(k)}$ corresponds to the security game in which all odd-level honest nodes have the same underlying identity, and all even-level nodes have the same underlying identity. $\mathcal{A}$'s view is independent on the anonymity bit in this hybrid

because the challenge node provides a chain that amounts to the same two keys signing each other over and over, so it's distributed the same way.

By the hybrid argument, $\mathcal{A}$'s view is therefore independent on the anonymity bit in hybrid $\mathcal{H}_0$, which corresponds to the real single-authority security game.