Delegatable Anonymous Credentials From Mercurial Signatures With Stronger Privacy

Scott Griffy[1], Anna Lysyanskaya[1], Omid Mir[2], Octavio Perez Kempner[3], and Daniel Slamanig[4]

[1] Brown University, {scott_griffy,anna_lysyanskaya}@brown.edu
[2] AIT Austrian Institute of Technology, omid.mir@ait.ac.at
[3] NTT Social Informatics Laboratories, octavio.perezkempner@ntt.com
[4] Universität der Bundeswehr München, daniel.slamanig@unibw.de

Abstract. Delegatable anonymous credentials (DACs) enable a root issuer to delegate credential-issuing power, allowing a delegatee to take a delegator role. To preserve privacy, credential recipients and verifiers should not learn anything about intermediate issuers in the delegation chain. One particularly efficient approach to constructing DACs is due to Crites and Lysyanskaya (CT-RSA '19). In contrast to previous approaches, it is based on mercurial signatures (a type of equivalence-class signature), offering a conceptually simple design that does not require extensive use of zero-knowledge proofs. Unfortunately, current constructions of "CL-type" DACs only offer a weak form of privacy-preserving delegation: if an adversarial issuer (even an honest-but-curious one) is part of a user's delegation chain, they can detect when the user shows its credential. This is because the underlying mercurial signature schemes allows a signer to identify his public key in a delegation chain.

We propose CL-type DACs that overcome the above limitation based on a new mercurial signature scheme that provides adversarial public key class hiding which ensures that adversarial signers who participate in a user's delegation chain cannot exploit that fact to trace users. We achieve this introducing structured public parameters for each delegation level. Since the related setup produces critical trapdoors, we discuss techniques from updatable structured reference strings in zero-knowledge proof systems (Groth et al. CRYPTO '18) to guarantee the required privacy needs. In addition, we propose a simple way to realize revocation for CL-type DACs via the concept of revocation tokens. While we showcase this approach to revocation using our DAC scheme, it is generic and can be applied to any CL-type DAC system. Revocation is a vital feature that is largely unexplored and notoriously hard to achieve for DACs, thus providing revocation can help to make DAC schemes more attractive in practical applications.

Keywords: Anonymous credentials · delegatable credentials · mercurial signatures · revocation · public key class-hiding.

1 Introduction

Anonymous credentials (ACs) allow an authority (the issuer) to issue user credentials that can then be used for anonymous authentication. This primitive was envisioned by Chaum in [Cha85] and later technically realized by Camenisch and Lysyanskaya in [CL01]. Importantly, in an AC scheme, a verifier and a user (also called a "credential holder") engage in a showing (also called a "proof" or "presentation") which proves to the verifier that the user has a valid credential. The scheme is *anonymous* if a user can show their credential multiple times in an unlinkable fashion. Intuitively, anonymity means that after verifying the credentials of two users, an adversary should not be able to tell if the credentials are both from a single user or from two different users.

Delegatable anonymous credentials (DACs) were introduced by Chase and Lysyanskaya [CL06]. As the name suggests, DAC schemes allow a root issuer to delegate their credential-issuing power to other "intermediate" issuers. This delegation allows any intermediate issuer to issue credentials on behalf of the root issuer (or possibly, re-delegate their issuing power), creating a *delegation chain* between the root issuer, the intermediate issuers, and the credential holder. Belenkiy, Camenisch, Chase, Kohlweiss, Lysyanskaya and Shacham showed how to realize DACs for arbitrarily long delegation chains [BCC+09].

Delegation alleviates the burden on the root issuer without revealing the root issuer's secrets to any other issuer, similar to a key hierarchy in a public key infrastructure (PKI) system. Unlinkability of DACs ensures the anonymity of credential holders, as well as the anonymity of any issuers who participated in that credential's delegation chain. The anonymity of intermediate issuers implies that given the showing of two credentials, an adversarial verifier cannot determine if they were issued by the same intermediate issuer or different intermediate issuers. Hiding the intermediate issuer is important for a DAC scheme as revealing the identities of these intermediate issuers might reveal information about the end user. The root issuer is always identified in a showing as the verifier must trust some key for unforgeability.

An important property when practically using (D)ACs is non-transferability. This property ensures that users cannot easily share their credentials with other users. One way of providing this is to ensure that a user cannot share one of her credentials without sharing all of her credentials (and corresponding secrets). This is known as the "all-or-nothing" approach [CL01] and should disincentivize sharing of credentials by users' fear of losing control over their credentials. Another feature that is particularly important for the practical application of (delegatable) anonymous credentials is revocation. Unfortunately, this property is often neglected. It is quite clear that when preserving user's privacy, standard approaches to recovation known from classical PKI schemes do not work. While there are various different approaches to revocation of anonymous credentials [CKS10, CKS09, DHS15, CKL+16, CL01, BDG+23, CL02], in the delegatable setting this seems much harder to achieve and the topic is largely unexplored [AN11].

1.1 Previous Work on DAC and Motivation for our Work

As in the recent work of Mir, Slamanig, Bauer, and Mayrhofer [MSBM23], we are particularly interested in developing practical DAC schemes. For a broader understanding, readers are directed to their comprehensive overview. The first practical DAC was proposed by Camenisch, Drijvers, and Dubovitskaya [CDD17], but unfortunately they do not support an anonymous delegation phase. This, however is a crucial privacy requirement. The DACs by Blömer and Bobolz [BB18] as well as [MSBM23] represent two relevant and efficient DAC candidates as they have anonymous delegations and additionally, compact credentials. Unfortunately, [MSBM23] does not provide the important property of non-transferability, and for both [BB18] and [MSBM23] the delegated credential is distributed independently of any of the previous delegators. Consequently, it seems very hard to efficiently achieve revocation of delegators for those schemes.

Crites and Lysyanskaya [CL19] came up with a simple architecture (which we will call "CL-type DAC") for delegatable anonymous credentials that uses *mercurial signatures* (MS). These CL-type DACs bring the use of equivalence class signatures, extensively used in anonymous credentials [HS14, DHS15, FHS19, HS21, CLPK22, MSBM23], to the DAC setting (with numerous follow up works [CL21, MBG+23, PM23, NKTA24] on various aspects). In CL-type DACs, the "links" in a delegation chain are signatures; this chain includes the root's signature on the first intermediate issuer's public key; then for $i \geq 1$, the i^{th} intermediary's signatures on the $i+1^{st}$ intermediary's public key, and finally the last intermediate issuer's signature on the credential holder's public key. In order to ensure unlinkability, mercurial signatures allow randomization of both the signer's public key to an equivalent but unlinkable public key, and the randomization of the message to an equivalent but unlinkable message. As an example delegation chain in a CL-type DAC, would first require a root (or certification) authority (CA) which holds a signing key of an MS scheme and to issue a credential to a user, Alice. To do this, the root authority produces a signature $\sigma_{CA,A}$ on an MS public key pk_A of Alice. By demonstrating knowledge of the corresponding secret key to pk_A along with the root's signature on pk_A, Alice can authenticate herself. Now if Alice wants to delegate a credential to Bob, she uses her corresponding secret key to produce a signature $\sigma_{A,B}$ on Bob's MS public key pk_B, where the signature acts as a credential for Bob. Now Bob can authenticate himself by demonstrating knowledge of the corresponding secret key (to pk_B) and showing the signatures from both the root ($\sigma_{CA,A}$) and from Alice ($\mathsf{pk}_A, \sigma_{A,B}$). This principle can be applied to an arbitrarily long delegation chain. Now assume that Bob wants to show in a privacy-preserving way that he has been delegated a credential by CA. He can do this by demonstrating $(\mathsf{pk}_{CA}, \mathsf{pk}_A{}', \mathsf{pk}_B{}'), (\sigma'_{CA,A}, \sigma'_{A,B})$ where $\mathsf{pk}_A{}'$ and $\mathsf{pk}_B{}'$ are new representatives of the respective key classes (and by the properties of MS, they are unlinkable to the previous ones) and the signatures $(\sigma'_{CA,A}, \sigma'_{A,B})$ are adapted to the new messages and public

keys respectively (which are similarly unlinkable). In the concrete CL construction [CL19], the MS scheme works in a bilinear group setting where w.l.o.g. the public key of the CA lives in the second source group, $\mathbb{G}_2$, and the public key of Alice in $\mathbb{G}_1$. Consequently, since the public keys of the MS scheme on the next level need to live in the message space of the MS scheme of the previous level, one always needs to switch the groups for the MS scheme on every level, which is an important detail to keep in mind.

One important limitation of existing DAC approaches [CL19, CL21, PM23, MSBM23], besides not yet supporting revocation, is that they only satisfy a weak notion of privacy. In particular it is not possible to guarantee anonymity even in the case of an honest-but-curious delegator in the credential chain (or when the root authority and a single delegator on the delegation chain collaborate, in the case of [MSBM23]). In prior constructions of CL-type credentials [CL19, CL21, PM23] this is because public keys in these constructions are traceable (using the secret key) regardless of how they have been randomized. Thus, a malicious delegator can identify itself on a chain and break anonymity.

When it comes to revocation in DAC, the only work so far is by Acar and Nguyen [AN11], which is based on the generic DAC template in [BCC+09] from randomizable NIZK proofs and in addition uses homomorphic NIZK proofs. While this can be instantiated from the Groth-Sahai [GS08] proof system, this is not very attractive from a practical perspective due to significant costs. So having practical DACs with revocation is an open problem.

Consequently, there exists a gap in that we do not have practical DAC schemes with strong privacy guarantees that support practical revocation of delegators. Our aim is to close this gap.

Our work can be seen as the continuation of the research initiated by Crites and Lysyanskaya in [CL19], closing these existing gaps for practical DAC schemes. To do this, we create a mercurial signature scheme with a stronger privacy property called adversarial public key class-hiding. An overview of this approach is outlined in the technical overview in Section 1.3. Ensuring that maliciously created public keys in mercurial signatures are not traceable by their owners after being randomized has been an open problem since their introduction in [CL19]. A very recent work introduced the notion of interactive threshold mercurial signatures [NKTA24] to overcome said limitation, but it requires an interactive signing protocol that computes a signature from shares of a secret key that are distributed among parties. While such an approach is satisfactory for anonymous credentials and can also be used to distribute trust of the root authority in DAC schemes, it's unclear how it can be used to efficiently manage delegations. Instead, we introduce structured public parameters which we carefully glue over the delegation levels to enable strong privacy features (and without requiring any interaction). Since the setup of these parameters also produces trapdoors that endanger privacy, we show how to overcome this problem by using techniques well-known from updatable structured reference string in zero-knowledge proof systems [GKM+18]. For revocation in DAC, we introduce a new and practical approach that is applicable to any CL-type DAC scheme.

1.2 Our Contributions

Subsequently, we summarize our contributions:

New mercurial signatures. First, as our core building block, we define a new flavor of a mercurial signature scheme which satisfies a stronger class hiding property, namely *adversarial public-key class hiding* (APKCH). Unlike in the mercurial signature definition of Crites and Lysyanskaya [CL19], here the adversary comes up with a public key and signs a message of its choice; the challenge for an adversary is to distinguish between a randomized version of his own public key and signature and a fresh, unrelated public key and a signature on the same message under that fresh public key. We give a construction of an APKCH signature scheme in the generic group model. Adversarial public-key class hiding is sufficient to construct a DAC scheme with strong privacy.

New technique for revocation in DAC. We introduce a new revocation approach for DACs. The basic idea is that a revocation authority maintains a public deny list, which verifiers can use to ensure that a credential shown to them does not contain any revoked delegators. Thereby any user who wants to receive a credential or obtain delegation rights (while still supporting later revocation) must

first register their key with the revocation authority. This registration is anonymous and neither a verification of the user's identity is needed nor a proof of knowledge of their key needs to be performed. This gives a simple privacy-preserving way for revocation in DAC, can be used for any CL-type DAC and does so without resorting to heavy tools such as malleable NIZKs as done in [AN11].

Model and instantiation of DAC with strong privacy and revocation. We introduce a conceptually simple model for revocable DAC schemes with strong privacy using game-based security definitions. We believe that this notion is easier to use than the simulation-based security notions provided in [AN11]. Then, using our mercurial signature scheme with $\ell = 2$, we construct a conceptually simple DAC scheme with delegator revocation and without requiring extensive use of zero-knowledge proofs. We stress that this gives a CL-type DAC scheme, where privacy holds even when the adversary is allowed to corrupt the root and all delegators simultaneously.

1.3 Technical Overview

Public keys in the mercurial signature from [CL19] are ℓ-length vectors of group elements and are constructed as $\mathsf{pk} = \{\hat{X}_i\}_{i\in[\ell]} = \{\hat{P}^{x_i}\}_{i\in[\ell]}$ where the secret key is $\mathsf{sk} = \{x_i\}_{i\in[\ell]} \in (\mathbb{Z}_p^*)^\ell$ and $\hat{P}$ is the generator the second source group of a bilinear pairing. Anyone can randomize a public key pk to a new representative of the equivalence class to get $\mathsf{pk}' = \mathsf{pk}^\rho = \{\hat{X}_i^\rho\}_{i\in[\ell]}$ for $\rho \in \mathbb{Z}_p$. Unfortunately, such public keys are immediately recognizable to an adversary who holds the corresponding secret key. For an adversary to recognize a public key, it suffices to exponentiate each element in the public key by the inverse of the corresponding element in the secret key and check that the result has the form: $\{\hat{P}^\rho\}_{i\in[\ell]}$ (a vector of equal elements).

One approach to overcome the above limitation is to ensure that each element in a public key is computed over a distinct generator of the group where the discrete logarithms between these generators are random and not known. For example, if we add a trusted setup to the scheme from [CL19]: $\mathsf{Setup}(1^\lambda) \to \mathsf{pp} = \{\hat{P}_1, \hat{P}_2, ..., \hat{P}_\ell\}$ where $\hat{P}_i = \hat{P}^{\hat{b}_i}$ for ℓ randomly sampled $\hat{b}_i$ scalars in $\mathbb{Z}_p$ and public keys are computed as $\mathsf{pk} = \{\hat{X}_i = \hat{P}_i^{x_i}\}_{i\in[\ell]}$ then it can be shown that under the DDH assumption that an adversary cannot distinguish a randomization of their public key from a freshly sampled key.

This appears promising, especially since these $\hat{P}_i$ values are all distinct and can be efficiently sampled in the ROM. But, if an honest user receives a public key, it is not immediately clear how to ensure that it wasn't created maliciously so that they are recognizable (e.g., ensure the adversary did not compute the public key independent of the public parameters as $\mathsf{pk} = \{\hat{P}^{x_i}\}_{i\in[\ell]}$ which would be recognizable). To ensure that maliciously created public keys are computed over these bases without the need for zero knowledge proofs, we add what we call "verification bases" to the public parameters. The verification bases are structured so that they can be paired with the key to prove that it was computed using the trusted public parameters. To accomplish this, we need to expand the size of the public key vectors to double their length (2ℓ). This extra half of the public key will be the result of exponentiating different bases in the public parameters with the same secret key as in the first half. Specifically, our public parameters (w.r.t. public keys) consist of $\hat{\mathbf{B}} = \{\hat{B}_i\}_{i\in[2\ell]} = \{\hat{P}^{\hat{b}_1}, \ldots, \hat{P}^{\hat{b}_\ell}, \hat{P}^{\hat{b}_1\hat{d}_1}, \ldots, \hat{P}^{\hat{b}_\ell\hat{d}_\ell}\}$ such that $\mathsf{dlog}_{\hat{B}_i}(\hat{B}_{\ell+i}) = \hat{d}_i$. We then include the verification bases in the public parameters which are computed as: $\mathbf{V} = \{V_i\}_{i\in[\ell]} = \{P^{v_1\hat{d}_1}, \ldots, P^{v_\ell\hat{d}_\ell}, P^{v_1}, \ldots, P^{v_\ell}\}$ (for randomly sampled scalars, $\{v_i\}_{i\in[\ell]}$) such that $\forall i \in [\ell], e(V_i, \hat{B}_i) = e(V_{\ell+i}, \hat{B}_{\ell+i})$. Then, key pairs are computed as $\mathsf{sk} = \{x_i\}_{i\in[\ell]} \leftarrow\!\!\$\ \mathbb{Z}_p$, $\mathsf{pk} = \{\hat{X}_i\}_{i\in[\ell]} = \{\hat{B}_i^{x_i}\}_{i\in[\ell]} \| \{\hat{B}_{\ell+i}^{x_i}\}_{i\in[\ell]}$. We can see now that $\forall i \in [\ell], e(V_i, \hat{X}_i) = e(V_{\ell+i}, \hat{X}_{\ell+i})$. Thus, a verifier can take this length 2ℓ public key and use the verification bases ($\mathbf{V}$) to verify it by computing these pairings. If these pairing equations hold, then, because the elements in the upper half of $\hat{\mathbf{B}}$ are the only elements in the second source group that are exponentiated with $\hat{d}_i$, the adversary must have computed the upper half of the public key with these bases. Through a similar argument, the lower half of the public key must be computed over the lower half of the public key bases in the public parameters. Thus, if these pairing equations hold for a public key, then randomizations of that public key are unrecognizable to the adversary. Because we need correlated randomness in the trapdoors (e.g. both V_i and $\hat{B}_{\ell+i}$ are computed using $\hat{d}_i$) we can no longer use the ROM to generate these parameters and instead must use the common reference string (CRS) model.

We quickly run into a second problem as with these modified public keys as correctness falls apart when we attempt to sign messages. In [CL19], signatures are computed as $\sigma = (Z, Y, \hat{Y})$ with

$Z = (\prod_{i\in[\ell]} M_i^{x_i})^y$, $Y = P^{1/y}$, and $\hat{Y} = \hat{P}^{1/y}$. Verification is done via a pairing product equation: $e(Z, \hat{Y}) = \prod_{i\in[\ell]} e(M_i, \hat{X}_i)$, which will not verify with the new structure of public keys that we've introduced in this section. Therefore, we expand the message space to vectors of 2ℓ elements instead of ℓ elements and include $\forall i \in [\ell], P_i = P^{\hat{b}_i}$ in the public parameters. Messages then have the form: $M = \{P^{m_i}\}_{i\in[\ell]} || \{P_i^{m_i}\}_{i\in[\ell]}$ for some vector $\{m_i\}_{i\in[\ell]} \in \mathbb{Z}_p^\ell$. In this modified scheme, the structure of Y and $\hat{Y}$ remains unchanged, but we then use the upper half of the message vector in our Sign function and the lower half of the message vector in our Verify function. This modification leads to a correct verification, now given by: $e(Z, \hat{Y}) = e(P, \hat{P})^{\sum_{i\in[\ell]} m_i x_i \hat{b}_i} = \prod_{i\in[\ell]} e(M_i, \hat{X}_i)$. We also have to add more structure to achieve a signature that yields DAC as the lower half of the message (which will be another public key in a DAC credential chain) is recognizable. We add this extra structure in Section 3.

Constructing a strongly private DAC. As previously mentioned, [CL19] works by alternating the use of two signature schemes so that even delegation levels are signed with one of the schemes and the odd ones with the other. This way, the message space in one of the schemes is the public key space of the other.

Our approach is to build a structured random string so that each scheme can sign public keys for the next level of the credential chain using unique blinding factors taken from the CRS for each level. This poses a challenge as we need to correlate the structure of both schemes for messages and public keys. To this end, the keys used in our scheme are twice the size of the keys in [CL19]. Fortunately, for CL-type DACs $\ell = 2$ and typical applications that use delegation do not require long delegation chains (e.g., driving licenses or official identity documents), making this approach entirely practical. To illustrate it, we consider a DAC scheme for $L = 3$. We can generate the public parameters, $\mathsf{pp} = \{P^{\hat{b}_0}, P^{\hat{b}_1}\}$, $\mathsf{pp}' = \{\hat{P}^{\hat{b}'_0}, \hat{P}^{\hat{b}'_1}, \hat{P}^{\hat{b}_0\hat{b}'_0}, \hat{P}^{\hat{b}_1\hat{b}'_1}\}$, $\mathsf{pp}^* = \{P^{\hat{b}^*_0}, P^{\hat{b}^*_1}, P^{\hat{b}'_0\hat{b}^*_0}, P^{\hat{b}'_1\hat{b}^*_1}\}$. We can see that the bases from pp and pp' have a structure that satisfies: $e(P^{\hat{b}_i}, \hat{P}^{\hat{b}'_i}) = e(P, \hat{P}^{\hat{b}_i\hat{b}'_i})$ and similar for pp' and pp^*. Hence, such public parameters can be used to build public keys for the credential chain as: $\mathsf{pk} = \{P^{\hat{b}_0 x_0}, P^{\hat{b}_1 x_1}\}$, $\mathsf{pk}' = \{\hat{P}^{\hat{b}'_0 x'_0}, \hat{P}^{\hat{b}'_1 x'_1}, \hat{P}^{\hat{b}_0\hat{b}'_0 x'_0}, \hat{P}^{\hat{b}_1\hat{b}'_1 x'_1}\}$, $\mathsf{pk}^* = \{P^{\hat{b}^*_0 x^*_0}, P^{\hat{b}^*_1 x^*_1}, P^{\hat{b}'_0\hat{b}^*_0 x^*_0}, P^{\hat{b}'_1\hat{b}^*_1 x^*_1}\}$. It follows from inspection that if we use $\mathsf{sk} = \{x_0, x_1\}$ to sign the third and fourth elements of pk', the signature will verify using the first and second elements from pk'. Similarly, if we use $\mathsf{sk}' = \{x'_0, x'_1\}$ to sign the third and fourth elements in pk^*, the signature will verify under the first half of pk' with the first half of pk^* as the message. Because these trapdoors are shared across schemes, we need the security properties of our signature scheme to hold when multiple correlated copies of the scheme are generated. We describe this requirement of our signature scheme further in Sec. 3.3 where we present the above example with more detail in Fig. 5.

Removing trust in the parameter generation. As it is apparent from our above discussion, the generation of parameters in setup involves a number of exponents that must not be available to any party. Putting trust in the party running this setup to discard these values is typically not desirable, especially in a DAC setting where there are numerous parties involved. One way to deal with this issue is to run specific multi-party computation protocols to generate the parameters [BCG+15, BGG19], e.g., running distributed key generation protocols. In order to avoid interaction among many parties, one particularly appealing approach was proposed by Groth et al. [GKM+18] in the context of zk-SNARKs, i.e., so called updatable reference strings. This means that some (potentially malicious) party can generate a reference string and any (potentially malicious) party in the system can update the reference string. Thereby every party outputs a proof that the computation was performed honestly and when the chain of proofs from the generation until the last update of the reference string verifies and at least one of the involved parties is honest, it is ensured that no one knows the trapdoors. Since this process can be done strictly sequentially this seems much more interesting for practical application, as for instance demonstrated by the powers of tau ceremony recently run by Ethereum[5], with around 95k contributors (cf. [NRBB24]). We note that in our case this can be done very efficiently using Fiat-Shamir transformed Schnorr proofs for discrete logarithm relations. In Appendix A.1 we present the concrete relations for the updates. In brief, the costs are 5ℓ Pedersen commitments for the trapdoors, 8ℓ group elements for the Schnorr proofs for the base group elements in pp and 10ℓ $\mathbb{Z}_p$ elements for the Schnorr proofs which include the trapdoors. Concretely,

[5] `https://github.com/ethereum/kzg-ceremony-specs`

for $\ell = 2$ this amounts to 26 group elements and 20 $\mathbb{Z}_p$ elements per level (where for usual applications $L \leq 5$) being several orders of magnitude smaller than the one run by Ethereum. We also note that the computation and verification of these Schnorr proofs is highly efficient.

Revoking intermediate issuers and users in a DAC scheme. Conceptually (but not technically) our approach to revocation shares similarities with verifier local revocation kown from group signatures [BS04]. In particular, to revoke these credentials, we generate revocation tokens that verify with a given public key and can later be provided to a trusted revocation authority (TRA). The TRA adds these tokens to a deny list. To achieve this, the TRA first creates the ephemeral secret and public keys. The TRA then registers the user by signing the user's public keys with the corresponding ephemeral secret key. This forms a signature chain similar to the mercurial scheme described in [CL19].

When a user needs to prove their credentials, they present a revocation token for each public key in their chain. Since this revocation method mirrors the credential chain approach in [CL19], i.e., mercurial signatures on public keys, the tokens can be randomized to maintain user anonymity during the presentation.

To revoke a user or issuer in a credential chain (perhaps if the credential chain is used to perform some illegal action) these revocation tokens can be supplied from the verifier to the TRA who can then look through all the secret ephemeral keys they generated to recognize the credential chain and add the respective ones to a deny list.

Later, any verifier can verify the TRA's signature in the revocation token and iterate through the deny list, using each secret key to attempt to match each public key in the chain. More specifically, the verifier exponentiates the ephemeral public key in the revocation token with the inverses of the secret keys in the deny list (as described earlier in Sec. 1.3). If a match is found, the verifier can confirm that the credential has been revoked (cf. Sec. 4 for details).

2 Background

Notation. We use $[\ell]$ to denote the set, $\{1, 2, \ldots, \ell\}$. We use the notation $x \in \mathsf{Func}$ to mean that x is a possible output of the function, Func. When drawing multiple values from a set, we may omit notation for products of sets, e.g. $(x, y) \in \mathbb{Z}_p$ is the same as $(x, y) \in (\mathbb{Z}_p)^2$ where only the latter is formally correct. For a map from the set Z to the set S, $m : Z \to S$, we will denote $m[i] \in S$ as the output of the map in S with input $i \in Z$. We use bold font to denote a vector (e.g. $\mathbf{V}$). For brevity, we will sometimes denote the elements in a vector as $\mathbf{V} = \{V_i\}_{i \in [\ell]} = \{V_1, \ldots, V_\ell\}$. For a vector, $\mathbf{V} = \{V_1, \ldots, V_\ell\}$, of group elements, we denote the exponentiation of each element by a scalar $(\rho \in \mathbb{Z}_p)$ with the notation: $\mathbf{V}^\rho = \{V_1^\rho, \ldots, V_\ell^\rho\}$. We use "wildcards" $(*)$ in equations. For example, the equation $(a, b) = (a', *)$, holds if $a = a'$ no matter what the value of b is. By $(m, *) \in S$ we mean there is a tuple in the set S such that the first element of the tuple is m and the second element is another value which could be anything. $\{(m, *) \in S : A(m)\}$ is the set of all tuples in S with m as their first element meeting condition A. For two distributions, A and B, we use the notation, $A \sim B$, to denote that they are computationally indistinguishable.

2.1 Bilinear Pairings

A bilinear pairing is a set of cyclic groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of prime order p, along with a pairing function, e (where $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$) which preserves structure. We call $\mathbb{G}_T$ the "target group" and call $\mathbb{G}_1$ and $\mathbb{G}_2$ the first and second "source groups" respectively. In this work, we use Type III pairings, which means that there is no efficient, non-trivial homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. The pairing function is efficiently computable and has a bilinearity property such that if $\langle P \rangle = \mathbb{G}_1$ and $\langle \hat{P} \rangle = \mathbb{G}_2$, then for $a, b \in \mathbb{Z}_p^*$, $e(P^a, \hat{P}^b) = e(P, \hat{P})^{ab}$. In our pairing groups, the Diffie-Hellman assumptions hold in the related groups, such that for $a, b, c \leftarrow\!\$\ \mathbb{Z}_p$, $(P^a, P^b, P^{ab}) \sim (P^a, P^b, P^c)$. Also, given (P^a, P^b) it is difficult to compute P^{ab}. We'll use hats to denote elements in the second source group, e.g. $\hat{X} \in \mathbb{G}_2, X \in \mathbb{G}_1$. We also use the generic group model in the bilinear pairing setting [Sho97].

2.2 Mercurial Signatures

The original scheme from [CL19] comprises the following algorithms: Setup, KeyGen, Sign, Verify, ConvertPK, ConvertSK, ConvertSig, and ChangeRep. The scheme is parametrized by a length, ℓ, which determines the upper bound on the size of messages that can be signed. A mercurial signature scheme is parameterized by equivalence relations for the message, public key, and secret key spaces: $\mathcal{R}_M$, $\mathcal{R}_{\mathsf{pk}}$, $\mathcal{R}_{\mathsf{sk}}$. These relations form *equivalence classes* for messages and keys and define exactly how messages and signatures can be randomized such that their corresponding signatures can correctly be updated to verify with the updated keys and messages. Allowing the keys and messages to be randomized is what gives this signature scheme its privacy-preserving properties. In this work, we introduce auxiliary algorithms to verify the correctness of messages and public keys with respect to the scheme parameters. These are VerifyMsg and VerifyKey, respectively. Thus, the syntax of mercurial signatures used in this work is given by:

- $\mathsf{Setup}(1^\lambda, 1^\ell) \to (\mathsf{pp}, td)$: Outputs public parameters pp, including parameterized equivalence relations for the message, public key, and secret key spaces: $\mathcal{R}_M$, $\mathcal{R}_{\mathsf{pk}}$, $\mathcal{R}_{\mathsf{sk}}$ and the sample space for key and message converters. This function also outputs a trapdoor (td) that can be used (in conjunction with the corresponding secret key) to recognize public keys.
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{sk})$: Generates a key pair.
- $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, M) \to \sigma$: Signs a message M with the given secret key.
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}, M, \sigma) \to (0 \text{ or } 1)$: Returns 1 iff σ is a valid signature for M w.r.t. pk.
- $\mathsf{ConvertPK}(\mathsf{pp}, \mathsf{pk}, \rho) \to \mathsf{pk}'$: Given a key converter ρ, returns pk' by randomizing pk with ρ.
- $\mathsf{ConvertSK}(\mathsf{pp}, \mathsf{sk}, \rho) \to \mathsf{sk}'$: Randomize a secret key such that it now corresponds to a public key which has been randomized with the same ρ (i.e. signatures from $\mathsf{sk}' = \mathsf{ConvertSK}(\mathsf{pp}, \mathsf{sk}, \rho)$ verify by the randomized $\mathsf{pk}' = \mathsf{ConvertPK}(\mathsf{pk}, \rho)$).
- $\mathsf{ConvertSig}(\mathsf{pp}, \mathsf{pk}, M, \sigma, \rho) \to \sigma'$: Randomize the signature so that it verifies with a randomized pk' (which has been randomized with the same ρ) and M, but σ' is otherwise unlinkable to σ.
- $\mathsf{ChangeRep}(\mathsf{pp}, \mathsf{pk}, M, \sigma, \mu) \to (M', \sigma')$: Randomize the message-signature pair such that $\mathsf{Verify}(\mathsf{pk}, M', \sigma') = 1$ (i.e., σ' and σ are indistinguishable) where M' is a new representation of the message equivalence class defined by M.
- $\mathsf{VerifyKey}(\mathsf{pp}, \mathsf{pk}) \to \{0, 1\}$: Takes a public key and verifies if it is well-formed w.r.t the public parameters pp.
- $\mathsf{VerifyMsg}(\mathsf{pp}, M) \to \{0, 1\}$: Takes a message and verifies if it is well-formed w.r.t the public parameters pp.

Along with defining the functions above, a mercurial signature construction also defines the equivalence classes that are used in the correctness and security definitions. In the construction of [CL19], relations and equivalence classes for messages, public keys, and secret key are defined as follows:

$$\mathcal{R}_M = \{(M, M') \in (\mathbb{G}_1^*)^\ell \times (\mathbb{G}_1^*)^\ell | \exists r \in \mathbb{Z}_p^* \text{ s.t. } M' = M^r\}$$

$$\mathcal{R}_{\mathsf{pk}} = \{(\mathsf{pk}, \mathsf{pk}') \in (\mathbb{G}_1^*)^\ell \times (\mathbb{G}_1^*)^\ell | \exists r \in \mathbb{Z}_p^* \text{ s.t. } \mathsf{pk}' = \mathsf{pk}^r\}$$

$$\mathcal{R}_{\mathsf{sk}} = \{(\mathsf{sk}, \mathsf{sk}') \in (\mathbb{Z}_p^*)^\ell \times (\mathbb{Z}_p^*)^\ell | \exists r \in \mathbb{Z}_p^* \text{ s.t. } \mathsf{sk}' = r \cdot \mathsf{sk}\}$$

Equivalence classes are denoted as $[M]_{\mathcal{R}_M}$, $[\mathsf{pk}]_{\mathcal{R}_{\mathsf{pk}}}$, $[\mathsf{sk}]_{\mathcal{R}_{\mathsf{sk}}}$ for messages, public keys, and secret keys respectively, such that: $[M]_{\mathcal{R}_M} = \{M' : (M, M') \in \mathcal{R}_M\}$, $[\mathsf{pk}]_{\mathcal{R}_{\mathsf{pk}}} = \{\mathsf{pk}' : (\mathsf{pk}, \mathsf{pk}') \in \mathcal{R}_{\mathsf{pk}}\}$, $[\mathsf{sk}]_{\mathcal{R}_{\mathsf{sk}}} = \{\mathsf{sk}' : (\mathsf{sk}, \mathsf{sk}') \in \mathcal{R}_{\mathsf{sk}}\}$. Effectively, this means that two messages (M, M') are in the same equivalence class if there exists a randomizer, $\mu \in \mathcal{MC}$, such that $M' = M^\mu$ with a similar definition for public keys and secret keys. Because of the properties of equivalence classes (reflexivity, symmetry, and transitivity), the following relations hold: $[M]_{\mathcal{R}_M} = [M']_{\mathcal{R}_M}$ iff $(M, M') \in \mathcal{R}_M$, $[\mathsf{pk}]_{\mathcal{R}_{\mathsf{pk}}} = [\mathsf{pk}']_{\mathcal{R}_{\mathsf{pk}}}$ iff $(\mathsf{pk}, \mathsf{pk}') \in \mathcal{R}_{\mathsf{pk}}$, and $[\mathsf{sk}]_{\mathcal{R}_{\mathsf{sk}}} = [\mathsf{sk}']_{\mathcal{R}_{\mathsf{sk}}}$ iff $(\mathsf{sk}, \mathsf{sk}') \in \mathcal{R}_{\mathsf{sk}}$.

Besides the usual notions for correctness and unforgeability, security of mercurial signatures requires message class-hiding, origin-hiding and public key class-hiding. We recall the original definitions and provide some intuition.

Definition 1 (Correctness [CL19]). *A mercurial signature for parameterized equivalence relations, $\mathcal{R}_{\mathcal{M}}$, $\mathcal{R}_{\mathsf{pk}}$, $\mathcal{R}_{\mathsf{sk}}$, message randomizer space, sample_{μ}, and key randomizer space, sample_{ρ}, is correct if for all parameters (λ, ℓ), $\forall(\mathsf{pp}, td) \in \mathsf{Setup}(1^{\lambda}, 1^{\ell})$, and $\forall(\mathsf{sk}, \mathsf{pk}) \in \mathsf{KGen}(1^{\lambda})$, the following holds:*

- ***Verification.*** $\forall M \in \mathcal{M}, \sigma \in \mathsf{Sign}(\mathsf{sk}, M) : \mathsf{Verify}(\mathsf{pk}, M, \sigma) = 1 \wedge \mathsf{VerifyMsg}(\mathsf{pp}, M) = 1 \wedge \mathsf{VerifyKey}(\mathsf{pp}, \mathsf{pk}) = 1$.
- ***Key conversion.*** $\forall \rho \in \mathsf{sample}_{\rho}, (\mathsf{ConvertPK}(\mathsf{pk}, \rho), \mathsf{ConvertSK}(\mathsf{sk}, \rho)) \in \mathsf{KGen}(1^{\lambda})$, $\mathsf{ConvertSK}(\mathsf{sk}, \rho) \in [\mathsf{sk}]_{\mathcal{R}_{\mathsf{sk}}}$, *and* $\mathsf{ConvertPK}(\mathsf{pk}, \rho) \in [\mathsf{pk}]_{\mathcal{R}_{\mathsf{pk}}}$.
- ***Signature conversion.*** $\forall M \in \mathcal{M}, \sigma, \rho \in \mathsf{sample}_{\rho}, \sigma', \mathsf{pk}'$ *s.t* $\mathsf{Verify}(\mathsf{pk}, M, \sigma) = 1$, $\sigma' = \mathsf{ConvertSig}(\mathsf{pk}, M, \sigma, \rho)$, *and* $\mathsf{pk}' = \mathsf{ConvertPK}(\mathsf{pk}, \rho)$, *then* $\mathsf{Verify}(\mathsf{pk}', M, \sigma') = 1$.
- ***Change of message representation.*** $\forall M \in \mathcal{M}, \sigma, \mu \in \mathsf{sample}_{\mu}, M', \sigma'$ *such that* $\mathsf{Verify}(\mathsf{pk}, M, \sigma) = 1$ *and* $(M', \sigma') = \mathsf{ChangeRep}(\mathsf{pk}, M, \sigma; \mu)$ *then* $\mathsf{Verify}(\mathsf{pk}, M', \sigma') = 1$ *and* $M' \in [M]_{\mathcal{R}_M}$.

Unforgeability rules out forgeries on the same equivalence class for messages that have been queried to the signing oracle and public keys as these "forgeries" are actually guaranteed to be computable by correctness. Thus, only forgeries on new equivalence classes are accepted as valid forgeries.

Definition 2 (Unforgeability [CL19]). *A mercurial signature scheme for parameterized equivalence relations $\mathcal{R}_{\mathcal{M}}$, $\mathcal{R}_{\mathsf{pk}}$, $\mathcal{R}_{\mathsf{sk}}$, is unforgeable if for all parameters (λ, ℓ) and all PPT adversaries $\mathcal{A}$, having access to a signing oracle, there exists a negligible function negl such that:*

$$\Pr\left[\begin{array}{r|r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda}, 1^{\ell}) & \mathsf{Verify}(pk^*, M^*, \sigma^*) = 1 \\ (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}) & \wedge\ [pk^*]_{\mathcal{R}_{\mathsf{pk}}} = [pk]_{\mathcal{R}_{\mathsf{pk}}} \\ (\mathsf{pk}^*, M^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(sk, \cdot)}(\mathsf{pk}) & \wedge\ \forall M \in Q, [M^*]_{\mathcal{R}_M} \neq [M]_{\mathcal{R}_M} \end{array}\right] \leq negl(\lambda)$$

Where Q is the list of messages that the adversary queried to the Sign *oracle.*

Message class-hiding provides unlinkability in the message space, and it's implied by the DDH assumption in the original scheme from [CL19].

Definition 3 (Message class-hiding [CL19]). *For all λ, ℓ and all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that:*

$$\Pr\left[\begin{array}{r|c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda}, 1^{\ell}) & \\ M_1 \leftarrow \mathcal{M}; M_2^0 \leftarrow \mathcal{M}; M_2^1 \leftarrow [M_1]_{\mathcal{R}_M} & b' = b \\ b \leftarrow\$ \{0,1\}; b' \leftarrow \mathcal{A}(\mathsf{pp}, M_1, M_2^b) & \end{array}\right] \leq \frac{1}{2} + negl(\lambda)$$

Importantly, converted signatures should look like freshly computed signatures in the space of all valid ones. This notion is captured with the *origin-hiding* definitions as shown below.

Definition 4 (Origin-Hiding for ConvertSig [CL19]). *A mercurial signature scheme is origin-hiding for* ConvertSig *if, given any tuple (pk, σ, M) that verifies, and given a random key randomizer ρ, $\mathsf{ConvertSig}(\sigma, \mathsf{pk}, \rho)$ outputs a new signature σ' such that σ' is a uniformly sampled signature in the set of verifying signatures, $\{\sigma^* | \mathsf{Verify}(\mathsf{ConvertPK}(\mathsf{pk}, \rho), M, \sigma^*) = 1\}$.*

Definition 5 (Origin-Hiding for ChangeRep [CL19]). *A mercurial signature scheme is origin-hiding for* ChangeRep *if, given any tuple (pk, σ, M) that verifies, and given a random message randomizer μ, $\mathsf{ChangeRep}(\mathsf{pk}, M, \sigma, \mu)$ outputs a new message and signature M', σ' such that M' is a uniform sampled message in the equivalence class of M, $[M]_{\mathcal{R}_M}$, and σ' is uniformly sampled verifying signature in the set of verifying signatures for M', $\{\sigma^* | \mathsf{Verify}(\mathsf{pk}, M', \sigma^*) = 1\}$.*

For anonymous credentials such as the attribute-based credential (ABC) scheme from [FHS19], the notion of message class-hiding is sufficient to provide unlinkability alongside origin-hiding. This is because in ACs the adversary doesn't know the Diffie-Hellman coefficients of the message vector that

is signed to produce a credential (these coefficients are only known to the honest user who created the message). In DAC's schemes from mercurial signatures the messages to be signed are public keys, which may be provided by the adversary. Since the adversary knows the corresponding secret key, achieving a class-hiding notion for public keys is much harder. The definition below only considers honestly generated keys for which the adversary doesn't know the secret key. This is similar to the case of message-class hiding but it clearly restricts applications.

Definition 6 (Public key class-hiding [CL19]). *For all λ, ℓ and all PPT adversaries $\mathcal{A}$, there exists a negligible function (negl) such that:*

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell); (\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{KGen}(\mathsf{pp}); \\ (\mathsf{pk}_2^0, sk_2^0) \leftarrow \mathsf{KGen}(\mathsf{pp}, \ell(\lambda)); \rho \leftarrow\$ (\mathsf{pp}); \\ \mathsf{pk}_2^1 = \mathsf{ConvertPK}(\mathsf{pk}_1, \rho); \mathsf{sk}_2^1 = \mathsf{ConvertSK}(\mathsf{sk}_1, \rho); \\ b \leftarrow\$ \{0,1\}; b' \leftarrow \mathcal{A}^{\mathsf{Sign}(sk_1,\cdot),\mathsf{Sign}(sk_2^b,\cdot)}(\mathsf{pp}, \mathsf{pk}_1, \mathsf{pk}_2^b)\end{array} \middle| \; b' = b \right] \leq \frac{1}{2} + negl(\lambda)$$

Mercurial Signature Construction CL19 [CL19]. We review the mercurial signature construction from [CL19] in Fig. 1, so that the differences are clear when we present our construction in Sec. 3.

$\mathsf{Setup}(1^\lambda, 1^\ell) \rightarrow (\mathsf{pp})$: Generate a description of a cryptographic bilinear pairing, BP. Output $\mathsf{pp} = BP$.
$\mathsf{KGen}(\mathsf{pp})$: Sample $\mathsf{sk} = \{x_i\}_{i\in[\ell]} \leftarrow\$ \mathbb{Z}_p$ and let $\mathsf{pk} = \{\hat{X}_i\}_{i\in[\ell]}$ where $\forall i \in [\ell], \hat{X}_i = \hat{P}_i^{x_i}$.
$\mathsf{Sign}(\mathsf{sk}, M) \rightarrow \sigma$: Sample $y \leftarrow_\$ \mathbb{Z}_p^*$ and compute a signature: $\sigma = \left(Z = (\prod_{i=1}^{\ell} M_i^{x_i})^y, Y = P^{1/y}, \hat{Y} = \hat{P}^{1/y}\right)$.
$\mathsf{Verify}(\mathsf{pk}, M, \sigma) \rightarrow$ (0 or 1): Accept iff $\prod_{i=1}^{\ell} e(M_i, \hat{X}_i) = e(Z, \hat{Y})$, and $e(Y, \hat{P}) = e(P, \hat{Y})$.
$\mathsf{ConvertSig}(\mathsf{pk}, M, \sigma, \rho) \rightarrow \sigma'$: Sample $\psi \leftarrow\$ \mathbb{Z}_p$. Compute: $Z' = Z^{\psi\rho}$, $Y' = Y^{1/\psi}$, and $\hat{Y}' = \hat{Y}^{1/\psi}$. Output $\sigma' = (Z', Y', \hat{Y}')$.
$\mathsf{ConvertPK}(\mathsf{pk}, \rho) \rightarrow \mathsf{pk}'$: Compute: $\mathsf{pk}' = \mathsf{pk}^\rho$.
$\mathsf{ConvertSK}(\mathsf{sk}, \rho) \rightarrow \mathsf{sk}'$: Compute: $\mathsf{sk}' = \rho\mathsf{sk}$.
$\mathsf{ChangeRep}(M, \sigma, \mu) \rightarrow (M', \sigma')$: Sample $\psi \leftarrow\$ \mathbb{Z}_p$ and compute: $\sigma' = (Z' = Z^{\psi\mu}, Y' = Y^{1/\psi}\hat{Y}' = \hat{Y}^{1/\psi})$, valid for $M' = M^\mu$.

Fig. 1. CL19 Mercurial Signature Construction [CL19]

We note that a function ($\mathsf{RecognizePK}$ shown in Def. 7) can be added to this scheme to recognize any randomization of a public key given a secret key [GL24]. We use this in our DAC construction to tell if users have been revoked.

Definition 7 (Recognize function for CL19 public keys [GL24]).

- $\mathsf{RecognizePK}(\mathsf{pp}, \mathsf{sk}, \mathsf{pk}) \rightarrow \{0,1\}$ *Parse* pk *as* $\mathsf{pk} = \{\hat{X}_i\}_{i\in[\ell]}$. *Parse* $\mathsf{sk} = \{x_i\}_{i\in[\ell]}$. *Check if* $\forall i \in [\ell-1], \hat{X}_i^{x_{i+1}/x_i} = \hat{X}_{i+1}$.

3 New Mercurial Signature Construction

In this section we present our core mercurial signature scheme satisfying a stronger notion of adversarial public key class-hiding (APKCH), which will then build the basis for our DAC construction with strong privacy.

3.1 Modified Security Definitions

Subsequently, we present security definitions for our mercurial signature scheme that are modified (or added) when compared to previous work and before going into details we discuss their generality.

On the generality of our definitions. Since our main focus in this work is the construction of DAC, we include in our basic definitons (adversarial public-key class hiding and unforgeability) a "levels" parameter L, which tells the challenger how many correlated schemes to construct (i.e., how many levels there will be in the delegation chain of the DAC). In our definitions, after receiving the public parameters for every level, the adversary picks a level, i, and the challenger generates a public key for that level to complete the game with. This allows the reductions in our proofs to ensure that the DAC scheme appears correct to the adversary while reducing APKCH to the anonymity of the DAC scheme. To reduce to unforgeability, we need a similarly modified definition for unforgeability where the challenger generates a number of levels and the adversary chooses which level to continue the game with. Clearly, by setting $L = i = 1$ we obtain versions of the definitions for a standalone mercurial signature scheme. In our definitions, Setup outputs a set of correlated parameters of different signature schemes (pp) and we use pp_i to refer to the parameters that define level i.

First, we formalize the APKCH notion in Def. 8. In this definition, first the challenger generates the public parameters and a public key and gives these to the adversary. The adversary then constructs a message, a key pair and a signature and returns these to the challenger. The challenger then either randomizes the adversary's public key and signature or creates a new signature on the same message from a randomization of the challenger's key. The randomizers are drawn from the "key converter space", $\mathcal{KC}$, which is defined by the construction (commonly, $\mathbb{Z}_p^*$). The adversary is then challenged to determine if the returned key/signature pair is randomized from their own key/signature that they supplied, or if it is a signature from the randomized challenger key. Looking ahead, this property ensures that in a DAC scheme, an adversary cannot determine if they themselves provided a credential to the prover (user) or if another issuer created the credential.

Definition 8 (Adversarial public key class-hiding). *A mercurial signature, Γ, has adversarial public key class-hiding if for all parameters (λ, ℓ, L), the advantage of any PPT set of algorithms $\mathcal{A} = \{\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2\}$, (labeled as $\mathsf{Adv}^{\mathsf{APKCH}}_{\Gamma,\mathcal{A}}(\lambda)$) is negligible,*

$$\mathsf{Adv}^{\mathsf{APKCH}}_{\Gamma,\mathcal{A}}(\lambda) := \left|\Pr\left[\boldsymbol{Exp}^{\mathsf{APKCH},0}_{\Gamma,\mathcal{A}}(\lambda) = 1\right] - \Pr\left[\boldsymbol{Exp}^{\mathsf{APKCH},1}_{\Gamma,\mathcal{A}}(\lambda) = 1\right]\right|$$

where $\boldsymbol{Exp}^{\mathsf{APKCH},b}_{\Gamma,\mathcal{A}}(\lambda)$ is the experiment shown in Figure 2.

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell, 1^L)$
$(i, \mathsf{st}) \leftarrow \mathcal{A}_0(\mathsf{pp})$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(\mathsf{pp}_i)$;
$(\mathsf{pk}_{\mathcal{A}}, \sigma_{\mathcal{A}}, M, \mathsf{st}) \leftarrow \mathcal{A}_1^{\mathsf{Sign}(\mathsf{pp}_i, \mathsf{sk}, \cdot)}(\mathsf{pp}_i, \mathsf{pk}, \mathsf{st})$
$\sigma \leftarrow \mathsf{Sign}(\mathsf{pp}_i, \mathsf{sk}, M)$
$\rho_0 \leftarrow\$ \mathcal{KC}$; $\mathsf{pk}^0 \leftarrow \mathsf{ConvertPK}(\mathsf{pp}_i, \mathsf{pk}, \rho_0)$; $\sigma^0 \leftarrow \mathsf{ConvertSig}(\mathsf{pp}_i, \sigma, \rho_0)$
$\rho_1 \leftarrow\$ \mathcal{KC}$; $\mathsf{pk}^1 \leftarrow \mathsf{ConvertPK}(\mathsf{pp}_i, \mathsf{pk}_{\mathcal{A}}, \rho_1)$; $\sigma^1 \leftarrow \mathsf{ConvertSig}(\mathsf{pp}_i, \sigma_{\mathcal{A}}, \rho_1)$
if $\mathsf{Verify}(\mathsf{pp}_i, \mathsf{pk}_{\mathcal{A}}, \sigma_{\mathcal{A}}, M) = 1 \ \wedge \ \mathsf{VerifyMsg}(\mathsf{pp}_i, \mathsf{pp}, M) = 1 \ \wedge$
$\mathsf{VerifyKey}(\mathsf{pp}_i, \mathsf{pp}, \mathsf{pk}_{\mathcal{A}}) = 1$, **return** $\mathcal{A}_2^{\mathsf{Sign}(\mathsf{pp}_i, \mathsf{sk}, \cdot)}(\mathsf{pp}_i, \mathsf{pp}, \mathsf{st}, \mathsf{pk}^b, \sigma^b)$
else return $\mathcal{A}_2^{\mathsf{Sign}(\mathsf{pp}_i, \mathsf{sk}, \cdot)}(\mathsf{pp}_i, \mathsf{pp}, \mathsf{st}, \bot, \bot)$

Fig. 2. Adversarial public key class-hiding experiment $\mathbf{Exp}^{\mathsf{APKCH},b}_{\Gamma,\mathcal{A}}(\lambda)$.

Finally, we provide the unforgeability definition that also considers multiple levels in Def. 9.

Definition 9 (Unforgeability). *A mercurial signature scheme for parameterized equivalence relations $\mathcal{R}_{\mathcal{M}}$, $\mathcal{R}_{\mathsf{pk}}$, $\mathcal{R}_{\mathsf{sk}}$, is unforgeable if for all parameters (λ, ℓ, L) and all PPT adversaries $\mathcal{A}$, having access to a signing oracle, there exists a negligible function negl such that:*

$$\Pr\left[\begin{array}{r|l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell, 1^L) & \mathsf{Verify}(\mathsf{pp}_i, pk^*, M^*, \sigma^*) = 1 \\ (i, \mathsf{st}) \leftarrow \mathcal{A}_0(\mathsf{pp}) & \wedge\ [pk^*]_{\mathcal{R}_{\mathsf{pk}}} = [pk]_{\mathcal{R}_{\mathsf{pk}}} \\ (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(\mathsf{pp}_i); & \wedge \forall M \in Q, [M^*]_{\mathcal{R}_M} \neq [M]_{\mathcal{R}_M} \\ (\mathsf{pk}^*, M^*, \sigma^*) \leftarrow \mathcal{A}_1^{\mathsf{Sign}(\mathsf{pp}_i, sk, \cdot)}(\mathsf{pk}) & \end{array}\right] \leq negl(\lambda)$$

Where Q is the list of messages that the adversary queried to the Sign *oracle.*

3.2 Construction

In Fig. 3, we construct a mercurial signature (MS) scheme which in particular provides adversarial public key class-hiding (APKCH) as defined in Def. 8. We fix $L = 1$ for this construction and explain how we can set correlated parameters while still achieving APKCH in Sec. 3.3.

As in prior MS constructions, our equivalence classes will be parameterized by the public parameters consisting of a description of the bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, p, P, \hat{P})$. Unlike prior constructions, they will also be parameterized by several length-2ℓ vectors that are part of the public parameters of the system. Specifically, the public parameters will include the vectors $\mathbf{B} = (B_1, \ldots, B_{2\ell})$, $\hat{\mathbf{B}} = (\hat{B}_1, \ldots, \hat{B}_{2\ell})$, and $\hat{\mathbf{V}} = (\hat{V}_1, \ldots, \hat{V}_{2\ell})$, $\mathbf{V} = (V_1, \ldots, V_{2\ell})$. These public parameters have trapdoors which include $\mathbf{b} = (b_1, \ldots, b_\ell)$, $\hat{\mathbf{b}} = (\hat{b}_1, \ldots, \hat{b}_\ell)$, and $\hat{\mathbf{d}} = (\hat{d}_1, \ldots, \hat{d}_\ell)$ which are vectors in $\mathbb{Z}_p^\ell$ and sampled uniformly randomly by Setup. The vector $\mathbf{B}$ will be used to define the message space (and construct messages), while the vector $\hat{\mathbf{B}}$ defines the public key space and allows users to create valid public keys. These public parameters are structured based on the trapdoors, such that $\forall i \in [\ell], B_i = P^{b_i}$, $B_{\ell+i} = {B_i}^{\hat{b}_i}$, $\forall i \in [\ell], \hat{B}_i = \hat{P}^{\hat{b}_i}$ and $\hat{B}_{\ell+i} = \hat{B}_i^{\hat{d}_i}$. To verify that keys and messages are computed over these bases, we include the verification bases (shown above as $\hat{\mathbf{V}}$ and $\mathbf{V}$) in the public parameters which are constructed as: $\forall i \in [\ell], \hat{V}_i = \hat{P}^{\hat{v}_i b_i}$, $\hat{V}_{\ell+i} = \hat{P}^{\hat{v}_i}$, $\forall i \in [\ell], V_i = P^{v_i \hat{d}_i}$ and $V_{\ell+i} = P^{v_i}$. The vector $\hat{\mathbf{V}}$ is used to verify messages while the vector $\mathbf{V}$ is used to verify keys as described in Sec. 1.3. Structuring the parameters in this way ensures that our construction achieves adversarial public key class-hiding as discussed in Sec. 1.3 and defined in Def. 8. Let pp denote the public parameters.

Our message space will consist of vectors of 2ℓ dimensions over $\mathbb{G}_1$ that have certain structure determined by pp; i.e., not every 2ℓ-dimensional vector will be a valid message. Specifically, our message space, $\mathcal{M}^{pp,\ell}$ is defined as:

$$\mathcal{M}^{pp,\ell} = \{(M_1, \ldots, M_{2\ell}) \mid \exists \mathbf{m} = (m_1, \ldots, m_\ell) \in \mathbb{Z}_p^\ell \text{ such that } \forall 1 \leq i \leq \ell \;\; M_i = {B_i}^{m_i} \wedge M_{\ell+i} = {B_{\ell+i}}^{m_i}\}.$$

Note that, using pp, it is possible to verify that a 2ℓ-dimensional vector is in the message space. In our scheme, the public parameters will include extra bases $\hat{\mathbf{V}} = \{\hat{V}_1, \ldots, \hat{V}_{2\ell}\}$ to pair with messages to verify them, i.e. ensuring that $e(M_i, \hat{V}_i) = e(M_{i+\ell}, \hat{V}_{i+\ell}))$. Moreover, they include extra bases $\mathbf{V} = \{V_1, \ldots, V_{2\ell}\}$, to pair with public keys in the same manner, i.e. $e(V_i, \hat{X}_i) = e(V_{i+\ell}, \hat{X}_{i+\ell})$. We label messages as $\mathbf{M} = \{M_1, \ldots, M_{2\ell}\}$ and public keys as $\mathsf{pk} = \{\hat{X}_1, \ldots, \hat{X}_{2\ell}\}$. We are now ready to define our equivalence class over the message space, which is the same as in prior work [CL19]:

$$R_{\mathcal{M}}^{pp,\ell} = \{(\mathbf{M}, \mathbf{M}') : \exists \mu \in \mathbb{Z}_p \text{ s.t. } \mathbf{M}, \mathbf{M}' \in \mathcal{M}^{pp,\ell} \wedge \mathbf{M}' = \mathbf{M}^\mu\}.$$

Our public key space will be defined similarly to our message space, but defined over vectors $\hat{\mathbf{B}} = (\hat{P}^{\hat{b}_1}, \ldots, \hat{P}^{\hat{b}_\ell}, \hat{P}^{\hat{b}_1 \hat{d}_1}, \ldots, \hat{P}^{\hat{b}_\ell \hat{d}_\ell})$ included in pp as well. Like messages, our public key space is a strict subset of the space of 2ℓ-dimension vectors in $\mathbb{G}_2^{2\ell}$, defined below:

$$\mathcal{PK}^{pp,\ell} = \{(\hat{X}_1, \ldots, \hat{X}_{2\ell}) \mid \exists \mathbf{x} = (x_1, \ldots, x_\ell) \in \mathbb{Z}_p^\ell \text{ such that } \forall 1 \leq i \leq \ell \;\; \hat{X}_i = \hat{B}_i^{x_i} \wedge \hat{X}_{\ell+i} = \hat{B}_{\ell+i}^{x_i}\}.$$

We define our equivalence classes over the public key space (similarly to [CL19]):

$$R_{\mathcal{PK}}^{pp,\ell} = \{(\mathsf{pk}, \mathsf{pk}') : \exists \rho \in \mathbb{Z}_p \text{ s.t. } \mathsf{pk}, \mathsf{pk}' \in \mathcal{PK}^{pp,\ell} \wedge \mathsf{pk}' = \mathsf{pk}^\rho\}.$$

We will see in the construction that the related structure of these messages and public keys (i.e. the fact that they both use the values $\mathbf{b}$ and $\hat{\mathbf{b}}$) ensures that randomized keys and signatures are not linkable even when the adversary holds the secret key $\{x_1, \ldots, x_\ell\}$, while also ensuring that signatures still correctly verify.

Our secret key space takes up the entire space of ℓ-dimensional vectors in $\mathcal{SK}^{pp,\ell} = (\mathbb{Z}_p)^\ell$ and is defined identically to [CL19] as:

$$R_{\mathsf{sk}}^{pp,\ell} = \{(\mathsf{sk}, \mathsf{sk}') : \exists \rho \in \mathbb{Z}_p \text{ s.t. } \mathsf{sk}, \mathsf{sk}' \in \mathsf{sk}^{pp,\ell} \wedge \mathsf{sk}' = \rho \mathsf{sk}\}$$

In the sequel, when clear from the context, we will omit the superscript pp, ℓ. In our construction, the key and message converter spaces are $\mathcal{KC} = \mathbb{Z}_p^*$ and $\mathcal{MC} = \mathbb{Z}_p^*$.

Our construction. We are now ready to present our MS construction in Fig. 3 which achieves our APKCH definition for $L = 1$. In Sec. 3.3, we will discuss modifications to the public parameter generation, allowing the scheme's extension for constructing DAC. While verifying the message or public key is not required for unforgeability, we include it in the Verify function as it is needed for public key class-hiding and message class-hiding.

$\mathsf{Setup}(1^\lambda, 1^\ell) \to (\mathsf{pp})$: Sample $\{b_i, \hat{b}_i, \hat{d}_i, \hat{v}_i, v_i\}_{i\in[\ell]} \leftarrow\$ \mathbb{Z}_p$ and compute

$\mathbf{B} = \{B_i\}_{i\in[2\ell]}$, where $\forall i \in [\ell], B_i \leftarrow P^{b_i}, B_{\ell+i} \leftarrow P^{b_i\hat{b}_i}$

$\hat{\mathbf{B}} = \{\hat{B}_i\}_{i\in[2\ell]}$, where $\forall i \in [\ell], \hat{B}_i \leftarrow \hat{P}^{\hat{b}_i}, \hat{B}_{\ell+i} \leftarrow \hat{P}^{\hat{b}_i\hat{d}_i}$

$\hat{\mathbf{V}} = \{\hat{V}_i\}_{i\in[2\ell]}$, where $\forall i \in [\ell], \hat{V}_i \leftarrow \hat{P}^{\hat{v}_i\hat{b}_i}, \quad \hat{V}_{\ell+i} \leftarrow \hat{P}^{\hat{v}_i}$

$\mathbf{V} = \{V_i\}_{i\in[2\ell]}$, where $\forall i \in [\ell], V_i \leftarrow P^{v_i\hat{d}_i}, \quad V_{\ell+i} \leftarrow P^{v_i}$

Output: $\mathsf{pp} = (\mathbf{B}, \hat{\mathbf{B}}, \hat{\mathbf{V}}, \mathbf{V})$

$\mathsf{KGen}(\mathsf{pp})$: Sample $\mathsf{sk} = \{x_i\}_{i\in[\ell]} \leftarrow\$ \mathbb{Z}_p$ and let $\mathsf{pk} = \{\hat{X}_i\}_{i\in[2\ell]}$ where $\forall i \in [\ell], \hat{X}_i = \hat{B}_i^{x_i}, \hat{X}_{i+\ell} = \hat{B}_{\ell+i}^{x_i}$.

$\mathsf{VerifyKey}(\mathsf{pp}, \mathsf{pk})$: Accept iff $\forall i \in [\ell], e(V_i, \hat{X}_i) = e(V_{i+\ell}, \hat{X}_{i+\ell})$.

$\mathsf{VerifyMsg}(\mathsf{pp}, M)$: Accept iff $\forall i \in [\ell], e(M_i, \hat{V}_i) = e(M_{i+\ell}, \hat{V}_{i+\ell})$.

$\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, M) \to \sigma$: If $\mathsf{VerifyMsg}(\mathsf{pp}, M) = 1$, sample $y \leftarrow\$ \mathbb{Z}_p^*$ and compute a signature:

$\sigma = \left(Z = (\prod_{i=1}^{\ell} M_{\ell+i}^{x_i})^y, Y = P^{1/y}, \hat{Y} = \hat{P}^{1/y}\right)$.

$\mathsf{Verify}(\mathsf{pk}, M, \sigma) \to (0 \text{ or } 1)$: Accept iff $\mathsf{VerifyMsg}(\mathsf{pp}, M) = 1$, $\mathsf{VerifyKey}(\mathsf{pp}, \mathsf{pk}) = 1$, $\prod_{i=1}^{\ell} e(M_i, \hat{X}_i) = e(Z, \hat{Y})$, and $e(Y, \hat{P}) = e(P, \hat{Y})$.

$\mathsf{ConvertSig}(\mathsf{pp}, \mathsf{pk}, M, \sigma, \rho) \to \sigma'$: Sample $\psi \leftarrow\$ \mathbb{Z}_p$. Compute: $Z' = Z^{\psi\rho}$, $Y' = Y^{1/\psi}$ and $\hat{Y}' = \hat{Y}^{1/\psi}$. Output $\sigma' = (Z', Y', \hat{Y}')$.

$\mathsf{ConvertPK}(\mathsf{pp}, \mathsf{pk}, \rho) \to \mathsf{pk}'$: Compute: $\mathsf{pk}' = \mathsf{pk}^\rho$.

$\mathsf{ConvertSK}(\mathsf{pp}, \mathsf{sk}, \rho) \to \mathsf{sk}'$: Compute: $\mathsf{sk}' = \rho\mathsf{sk}$.

$\mathsf{ChangeRep}(\mathsf{pp}, M, \sigma, \mu) \to (M', \sigma')$: Sample $\psi \leftarrow\$ \mathbb{Z}_p$ and compute: $\sigma' = (Z' = Z^{\psi\mu}, Y' = Y^{1/\psi}\hat{Y}' = \hat{Y}^{1/\psi})$, valid for $M' = M^\mu$.

Fig. 3. Our Mercurial Signature Construction.

Theorem 10 (Correctness). *The mercurial signature construction in Fig. 3 is correct as described Def. 1.*

Theorem 11 (Unforgeability). *The mercurial signature construction in Fig. 3 meets the unforgeability definition in Def. 2 assuming that the mercurial signature construction in [CL19] is unforgeable in the generic group model.*

We prove Theorem 11 by noting that the CL19 construction [CL19] is unforgeable in the generic group model, and thus, by showing that our construction's unforgeability relies solely on the unforgeability of the construction of CL19, our construction is also unforgeable in the generic group model.

Theorem 12 (APKCH). *The mercurial signature construction in Fig. 3 meets the* APKCH *definition in 8 in the generic group model.*

Theorem 13 (Origin-hiding of signatures). *The mercurial signature construction in Fig. 3 meets the Origin-hiding of signatures definition in Def. 4.*

Theorem 14 (Origin-hiding of ChangeRep). *The mercurial signature construction in Fig. 3 meets the Origin-hiding of* ChangeRep *definition in Def. 5.*

Theorem 15 (Message class-hiding). *The mercurial signature construction in Fig. 3 meets the Message class-hiding definition in Def. 3.*

The proofs of theorems 13, 14, and 15, follow directly from those of CL19, and are thus omitted here. The proofs of unforgeability (Theorem 11) and APKCH (Theorem 12) are provided in Appendix B.

3.3 Extending our Construction to Multiple Levels

In a CL-type DAC scheme, we need chains of public keys that can sign each other. In [CL19], this is achieved by alternating the source groups of the mercurial signature scheme for each level in the chain. For example, to sign public keys in the highest level of the delegation chain L, if the public keys in level L are in source group $\mathbb{G}_2$, then, in the level $L-1$, a scheme with public keys in $\mathbb{G}_1$ will be used to sign the public keys of level L.

This approach works in [CL19] because the scheme is symmetric, meaning the public parameters are the same whether public keys are in $\mathbb{G}_2$ or $\mathbb{G}_1$. Unfortunately, our scheme is not symmetrical. Looking at the Setup function in Fig. 3, we see that if we split our message bases and public key bases into halves, $\mathbf{B} = \mathbf{B^l} \| \mathbf{B^u}$ and $\hat{\mathbf{B}} = \hat{\mathbf{B}}^{\mathbf{l}} \| \hat{\mathbf{B}}^{\mathbf{u}}$, the second half of the bases for messages includes the trapdoors $\hat{b}_i$, being formed as $\mathbf{B^u} = \{P^{b_i \hat{b}_i}\}_{i \in [\ell]}$. This trapdoor is included in the first half of the bases for public keys ($\hat{\mathbf{B}}^{\mathbf{l}} = \{\hat{P}^{\hat{b}_i}\}_{i \in [\ell]}$). But, the upper half of the bases for public keys includes the trapdoors, $\hat{d}_i$ ($\hat{\mathbf{B}}^{\mathbf{u}} = \{\hat{P}^{\hat{b}_i \hat{d}_i}\}_{i \in [\ell]}$). The trapdoors $\hat{d}_i$ are not seen in the lower bases for messages ($\mathbf{B^l} = \{P^{b_i}\}_{i \in [\ell]}$). Due to this asymmetry, we cannot simply invert the groups to start signing public keys from higher levels. At first glance, it appears we could fix this by setting $\hat{d}_i = b_i$ (thus allowing messages to be used to sign public keys by computing the signatures on the second half of the public key). Unfortunately, this solution would break the APKCH property of our scheme as computing public keys over $\hat{P}^{b_i \hat{b}_i}$ permits a recognition attack using the bases $P^{b_i \hat{b}_i}$. This forces us to choose a more involved solution.

We can solve this problem using the Setup function in Fig. 4. This function produces $L-1$ levels where the message space of each scheme is exactly the public key space of the subsequent scheme (with the equivalence classes matching as well). To better explain this solution (and simplify our proofs) we discuss the notion of "extending" schemes in this rest of this Section.

$\mathsf{Setup}(1^\lambda, 1^\ell, 1^L) \to (\mathsf{pp})$: Sample $\{\hat{b}_{i,j}, v_{i,j}\}_{i \in [\ell], j \in [L]} \leftarrow\$ \mathbb{Z}_p, \{\hat{d}_i\}_{i \in [\ell]} \leftarrow\$ \mathbb{Z}_p$ and compute the following bases:

$\hat{\mathbf{B}}_0 = \{\hat{B}_i\}_{i \in [2\ell]}$, where $\forall i \in [\ell], \hat{B}_i \leftarrow P^{\hat{b}_{i,0}}, \hat{B}_{\ell+i} \leftarrow P^{\hat{b}_{i,0}\hat{d}_i}$

$\mathbf{V}_0 = \{V_i\}_{i \in [2\ell]}$, where $\forall i \in [\ell], V_i \leftarrow \hat{P}^{v_{i,0}\hat{d}_i}, V_{\ell+i} \leftarrow \hat{P}^{v_{i,0}}$

For $j \in [L] \setminus \{0\}$, if $j \mod 2 = 0$, $G = \hat{P}, G' = P$ and if $j \mod 2 = 1$, $G = P, G' = \hat{P}$,

$\hat{\mathbf{B}}_j = \{\hat{B}_i\}_{i \in [2\ell]}$, where $\forall i \in [\ell], \hat{B}_i \leftarrow G^{\hat{b}_{i,j}}, \hat{B}_{\ell+i} \leftarrow G^{\hat{b}_{i,j}\hat{b}_{i,j-1}}$

$\mathbf{V}_j = \{V_i\}_{i \in [2\ell]}$, where $\forall i \in [\ell], V_i \leftarrow (G')^{v_{i,j}\hat{b}_{i,j-1}}, V_{\ell+i} \leftarrow (G')^{v_{i,j}}$

For $j \in [L-1]$: $\mathsf{pp}_j = \{\hat{\mathbf{B}}_{j+1}, \hat{\mathbf{B}}_j, \mathbf{V}_{j+1}, \mathbf{V}_j\}$ such that $\hat{\mathbf{B}}_j, \mathbf{V}_j$ are for keys and $\hat{\mathbf{B}}_{j+1}, \mathbf{V}_{j+1}$ are for messages.

Output $\mathsf{pp} = \{\mathsf{pp}_j\}_{j \in [L-1]}$.

Fig. 4. Parameter generation for multiple levels

We will use the terms "lower" and "higher" to refer to different levels of signature scheme that will be used to construct DAC. The root key is at level 0 which is the lowest level of the delegation chain and user's keys are messages in the $L-1$ (highest) level. In order to sign higher-level public keys with lower-level public keys, starting from our construction in Fig. 3, we need to create multiple levels of the signature scheme so that lower level public key bases can be used to sign public keys from higher levels scheme. To do this, we need to create a new scheme (with public keys in the opposite source group, $\mathbb{G}_1$) with similar structure as in our original scheme in Fig. 3. We recall that in this scheme the message bases and public key bases share the $\hat{b}_i$ trapdoor values as described in the above paragraph. This can be imagined as "extending" a scheme to lower levels. When extending a scheme to enable signatures on the public keys, we'll treat $\hat{d}_i$ as this shared value, setting $\hat{b}_i$ for the lower scheme to be equal to $\hat{d}_i$ in the higher scheme (remember, $\hat{d}_i$ is used in the upper half of the public key bases, $\hat{\mathbf{B}}$ in the public parameters). For example, say we have a higher scheme (with bases $\hat{\mathbf{B}} = \{\hat{B}_i\}_{i \in [2\ell]}$) with key pair $(\mathsf{sk}, \mathsf{pk})$, where $\mathsf{sk} = \{x_i\}_{i \in [\ell]}$, $\mathsf{pk} = \{\hat{X}_i\}_{i \in [2\ell]} = \{\hat{B}_i^{x_i}\}_{i \in [\ell]} \| \{\hat{B}_{\ell+i}^{x_i}\}_{i \in [\ell]}$, $\forall i \in [\ell], \hat{B}_i = \hat{P}^{\hat{b}_i}, \hat{B}_{\ell+i} = \hat{P}^{\hat{b}_i \hat{d}_i}$ and $\hat{b}_i$ and $\hat{d}_i$ are randomly sampled as a trapdoor of the public parameters. We can create a lower scheme (with bases $\hat{\mathbf{B}}' = \{(\hat{B}'_i)\}_{i \in [2\ell]}$) and key pair $(\mathsf{sk}', \mathsf{pk}')$ where $\mathsf{sk}' = \{x'_i\}_{i \in [\ell]}$, $\mathsf{pk}' = \{X'_i\}_{i \in [2\ell]} = \{(\hat{B}'_i)^{x'_i}\}_{i \in [\ell]} \| \{(\hat{B}'_{\ell+i})^{x'_i}\}_{i \in [\ell]}$ and $\forall i \in [\ell], (\hat{B}'_i) = P^{\hat{d}_i}, (\hat{B}'_{\ell+i}) = P^{\hat{d}_i \hat{d}'_i}$ and $\hat{d}'_i$ is randomly sampled as a trapdoor of the public parameters. We can now use this lower

level scheme to sign the keys in the higher level. We can see that if we form signatures as we did in Fig. 3, these signatures still verify. In Fig. 3, (if we swap the source groups) signatures are formed as $\sigma = (Z, Y, \hat{Y})$ where $Z = (\prod_{i\in[\ell]}(\hat{X}_{\ell+i})^{x'_i})^y$, $Y = \hat{P}^{1/y}$, $\hat{Y} = P^{1/y}$ and y is randomly sampled. We see that $e(\hat{Y}, Z) = e(P, \hat{P})^{\sum_{i\in[\ell]} \hat{d}_i \hat{b}_i x_i x'_i} = \prod_{i\in[\ell]} e(\hat{X}_i, X'_i)$ which means that this signature verifies. We provide Fig. 5 to make multi-level signature schemes more clear. In Fig. 5, we can see that when the lower level bases of levels 1 and 2 are paired together, they are equal to the pairing of the higher level bases of level 2, i.e. $e(\hat{B}_{2,i}, \hat{B}_{1,i}) = e(P^{\hat{b}_{2,i}}, \hat{P}^{\hat{b}_{1,i}}) = e(P, \hat{P})^{\hat{b}_{2,i}\hat{b}_{1,i}} = e(\hat{B}_{2,\ell+i}, \hat{P})$. This structure is what ensures that our signatures verify (as described in Section 1.3). We've pointed out this structure by highlighting $\hat{b}_{1,i}$ in orange and $\hat{b}_{2,i}$ in blue in Fig. 5. This relation holds for levels 2 and 3 as well. Note that in Fig. 5, the source groups for level 2 are flipped, meaning that the $\hat{B}_{2,i}$ elements are in $\mathbb{G}_1$ and the Z_2 element is in $\mathbb{G}_2$. For clarity, we keep the notation of the generators, P and $\hat{P}$, correct with respect to levels 1 and 3.

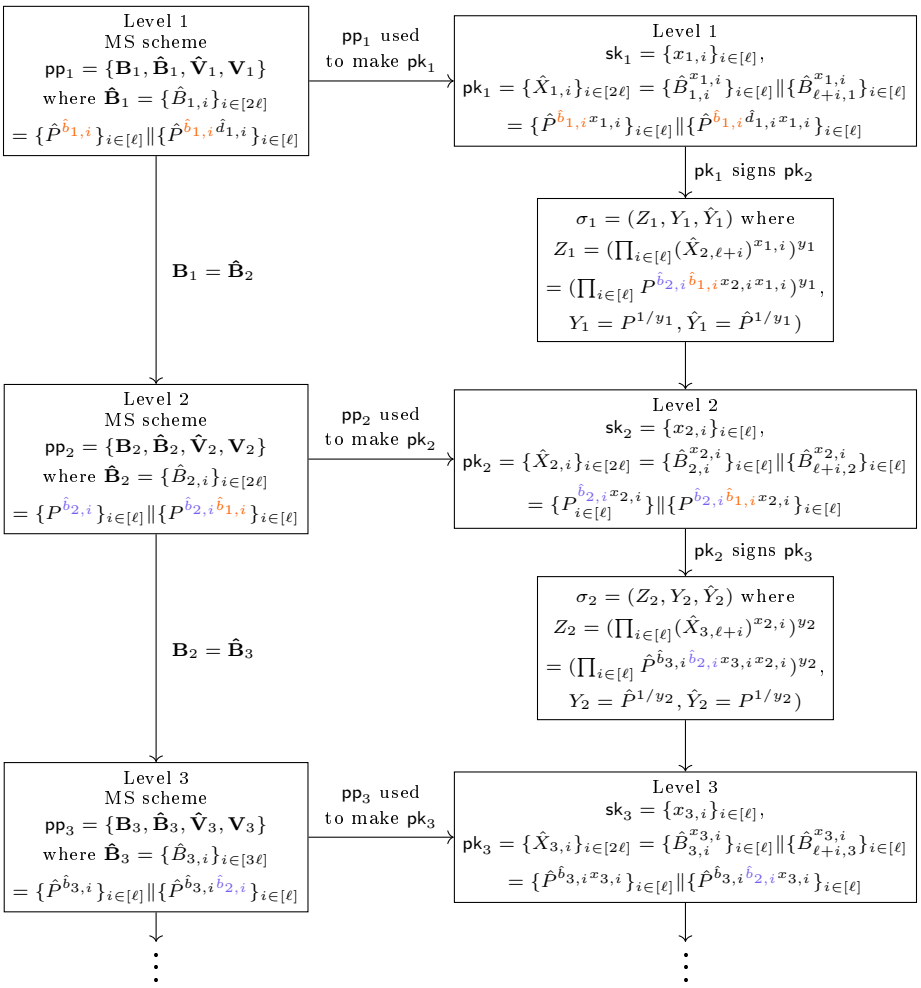

Fig. 5. A series of compatible mercurial signature schemes and a credential chain.

In Appendix A.2, we introduce the formal definition of extending parameters. This eases the readability of our proof as a generic group model proof for $L-1$ sets of public parameters simultaneously might be difficult to comprehend. Instead, we prove the APKCH security of a scheme for a single level that reveals enough secrets about the parameters so that the scheme can be extended to match the distribution of the parameters in Fig. 4. A simple hybrid arugment can then be used to prove that our multi-level scheme achieves APKCH as described in Def. 8. More specifically, in the

proof of APKCH in the full version, we use a Setup function (similar to Fig. 3) that also reveals $\mathbf{D} = \{D_i\}_{i\in[\ell]} = \{P^{\hat{d}_i}\}_{i\in[\ell]}$ such that if the keys for this scheme live in $\mathbb{G}_2$, then D_i is in $\mathbb{G}_1$. This allows a second setup to be run to create a lower level scheme that is compatible with the first scheme by computing: $\forall i \in [\ell], (\hat{B}'_i) = D_i, (\hat{B}'_{\ell+i}) = D_i^{\hat{d}'_i}$. We can see that this is exactly how the extended scheme computed their public parameters, $(\hat{B}'_{\ell+i})$ (explained earlier in this section) but now the second scheme does not know $\hat{d}_i$, which prevents attacks on class-hiding. Further, because $\mathbf{D}$ lives in $\mathbb{G}_1$ instead of $\mathbb{G}_2$, the adversary cannot use it to create malicious public keys that verify for the original scheme. While the real setup (in Fig. 4) will not reveal $\mathbf{D}$ to an adversary, it is important that the signatures retain their security properties even when this is revealed. Intuitively, this is because schemes built on top of a signature scheme requires some correlated structure. Revealing $D_i = P^{\hat{d}_i}$ in our security games ensures that this correlated structure cannot be leveraged to defeat the security of the schemes at other levels.

4 Delegatable Anonymous Credentials

In this section, we introduce a new DAC construction, showcasing advanced features as strengthened privacy, revocation capabilities, and non-transferability, all while preserving efficiency.

4.1 DAC Functionality

In contrast to non-revocable DAC schemes, our approach integrates a Trusted Revocation Authority (TRA) to efficiently revoke malicious users and maintain a deny list. We outline the high-level functionality of our DAC scheme in Def. 16. This consists of the algorithms: Setup which initializes the scheme, TKeyGen which generates the TRA's keys, RootKeyGen which generates the root's keys, UKeyGen which generates a user's or issuer's keys, RegisterUser which allows the TRA to distribute revocation tokens, and RevokeUser which allows the TRA to revoke users. The scheme also consists of the interactive protocols to issue and show credentials: (Issue $\leftrightarrow$ Receive) and (Prove $\leftrightarrow$ Verify). This scheme begins with a trusted Setup[6]. Then, the TRA generates an opener secret key and public key using TKeyGen. A root authority (who can be malicious for the sake of anonymity) generates the root key using RootKeyGen and distributes the root public key to users. A user who wishes to receive a credential runs UKeyGen and then interacts with the TRA to receive a revocation token by providing their public key to the TRA so that the TRA can run RegisterUser on it. Subsequently the user interacts with the root (or an issuer that the root has delegated to) using (Issue $\leftrightarrow$ Receive) and receives a credential (which includes their revocation token). The user then uses their credential and secret key in an interactive protocol (Prove $\leftrightarrow$ Verify) with any verifier. These verifications can occur at any level within $[L]$ (i.e. for some level, L' such that $L' \leq L$). The verifier can check if the user has been authorized and has not been revoked using the TRA's public key tpk. More specifically, the verifier receives a revocation token for each level in the credential chain from the credential presentation. If the verifier discovers that the prover was malicious through an out-of-band method, they can submit these tokens to the TRA. The TRA will then update their deny list (this deny list is included in the TRA public key for the sake of simplifying the presentation), preventing any future showings that include the user or issuer corresponding to the revocation token from being verified.

We note that our scheme supports a strong model for anonymity where the holder of the root key (colluding with intermediate issuers) cannot de-anonymize users. To model this, we allow the adversary in the anonymity game to choose the root public key along with the corruption of any users of their choice. This allows the adversary to choose any honest user's delegation path from a malicious root with all malicious delegations.

Definition 16 (Delegatable Anonymous Credentials). *A DAC scheme includes the following algorithms and protocols:*

– $\mathsf{Setup}(1^\lambda, 1^L) \to (\mathsf{pp}, td)$: Initializes the scheme, outputting public parameters and a trapdoor td.

[6] Note that as already discussed in practice this can be done by multiple parties in a sequential way by using ides from updatable common reference strings and only a single party among the set of all parties needs to be trusted.

- $\mathsf{TKeyGen}(\mathsf{pp}) \to (tsk, tpk)$: Takes pp and outputs a keypair (tsk, tpk) for the TRA. The tpk includes a deny list of revoked users and is continously updated.
- $\mathsf{RootKeyGen}(\mathsf{pp}) \to (\mathsf{sk}_{rt}, \mathsf{pk}_{rt})$: Generates a key pair used for the root key pk_{rt} (i.e. for level 0) which is trusted for integrity but does not need to be trusted for anonymity.
- $\mathsf{UKeyGen}(\mathsf{pp}, L') \to (\mathsf{sk}, \mathsf{pk})$: Generates a user's key pair for a specified level.
- $\mathsf{RegisterUser}(\mathsf{pp}, tsk, \mathsf{pk}) \to (tok)$: Creates a revocation token on the given public key so that the public key can be revoked later with a deny list.
- $(\mathsf{Issue}(\mathsf{sk}_I, \mathsf{cred}_I, L') \leftrightarrow \mathsf{Receive}(\mathsf{sk}_R, tok, L')) \to \mathsf{cred}_R$: An interactive protocol to receive a signature on a pseudonym. It is run between the two users distinguished by I for issuer or R for receiver. If $L' = 1$ (issuing from the root) then $\mathsf{cred}_I = \perp$.
- $(\mathsf{Prove}(\mathsf{sk}_P, \mathsf{cred}_P, L') \leftrightarrow \mathsf{Verify}(\mathsf{pk}_{rt}, L', tpk)) \to (b, \{tok_i\}_{i \in [L']})$: A user proves they know a credential on a pseudonym that verifies under the given root key, pk_{rt}. If the verification is successful, the verifier outputs 1 along with a list of revocation tokens for the prover and the chain of credentials. If the verification is unsuccessful, the verifier outputs 0.
- $\mathsf{RevokeUser}(\mathsf{pp}, tsk, tpk, tok) \to tpk'$: Takes in the TRA's key pair (tsk, tpk) as well as the token for a registered public key and outputs an updated public key tpk' that can be used to recognize any showings in which this public key is part of the chain. For security reasons, this can fail, outputting $\perp$. As an example, we want this function to fail if a malicious tok is provided.

4.2 DAC Security Definitions

In Fig. 6 we formally define the oracles used in our security games. Any formal outputs of oracles are received by the adversary and any modified internal state of the challenger is listed in the description. When calling interactive functions from the DAC scheme (such as $\mathsf{Prove}(\cdot) \leftrightarrow \mathsf{Verify}(\cdot)$), the challenger records the transcript of the interaction in addition to the output of the function. For example, in the VerifyCred oracle in Fig. 6, we have the challenger interact with the adversary using the Verify function, and in addition to outputting the result of the verification (b) and the list of revocation tokens, $\{tok_i\}_{i \in [L']}$, the protocol also outputs a transcript (τ) of the interaction between the prover and the verifier. Throughout the game, the challenger maintains some state to keep track of honest users and credentials that were given to the adversary. This global state is used in the unforgeability game. Specifically, the challenger keeps track of one set, $\mathsf{DEL}_{\mathcal{A}}$ to keep track of what has been delegated to the adversary. Moreover, the challenger initializes three maps to keep track of honest user state, SK holds user secret keys, CRED holds user credentials, and LVL records what level a user's credential is for. They are as follows: $\mathsf{SK} : \mathcal{H} \to \mathcal{SK}$, $\mathsf{CRED} : \mathcal{SK} \to \mathcal{CRED}$ and $\mathsf{LVL} : \mathcal{SK} \to [L]$, where $\mathcal{H}$ is the set of all honest user handles (which the adversary uses to refer to honest users), $\mathcal{SK}$ is the set of all secret keys, and $\mathcal{CRED}$ is the set of all credentials. The root key is included in SK with handle $id = 0$ where $\mathsf{LVL}[\mathsf{SK}[0]] = 0$ and $\mathsf{CRED}[\mathsf{SK}[0]] = \perp$. The challenger also keeps track of what keys have been added to the deny list with the list $\mathsf{SK}_{DL} \subset \mathcal{SK}$. At the start of any game, the challenger initializes all sets to the empty set and initializes all maps to be degenerate, such as mapping $\forall i \in \mathcal{H}, \mathsf{SK}[i] = \perp$. In the unforgeability game, we give the adversary access to all of the oracles. In the $\mathcal{O}^{\mathsf{CreateHonestUser}}$ oracle, the adversary specifies two users with one issuing a credential to the other. We initialize users at the same time that they are issued a credential to simplify the scheme. As an example use of this oracle, the adversary can specify $id_I = 0$ at the start of a game to have a credential be issued from the root. We do not allow the adversary to issue to a user multiple times, and thus if the specified user already exists when the adversary calls $\mathcal{O}^{\mathsf{CreateHonestUser}}$, then the challenger aborts. We allow the adversary to issue credentials to honest users using the $\mathcal{O}^{\mathsf{ReceiveCred}}$ oracle. In this oracle, the adversary specifies a user to receive a credential at a specified level. If the adversary was never issued a credential that would allow them to delegate this credential to the honest user, the challenger set a forgery flag in the global state (labeled *forgery*) which is checked in the unforgeability game. In the $\mathcal{O}^{\mathsf{IssueFrom}}$ oracle, the adversary receives a credential from an honest user and the challenger records which adversarial key received this credential at which level. In the $\mathcal{O}^{\mathsf{ProveTo}}$ oracle, the adversary acts as the verifier for a user. In the $\mathcal{O}^{\mathsf{RegisterUser}}$ oracle the adversary can receive a revocation token for one of their public keys. In the $\mathcal{O}^{\mathsf{RevokeUser}}$ oracle, the adversary can add a user to the deny list.

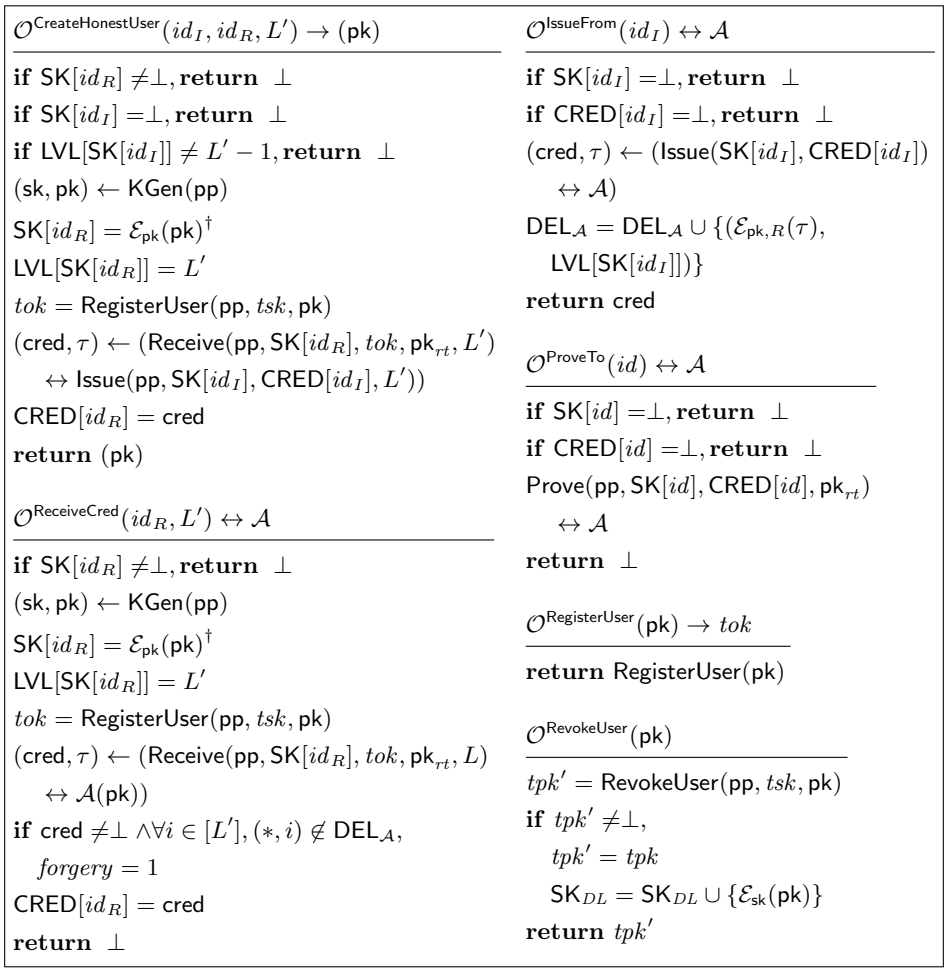

Fig. 6. Definition of Oracles.
† The oracle uses $\mathcal{E}_{\mathsf{pk}}$ to ensure $\mathsf{SK}[id]$ holds a canonical representation of the secret key.

We define correctness for our strongly private DAC scheme in Def. 17. If the probability for delegator issuance holds, we know that the scheme is correct for issuance to all levels, $1, ..., L$ as the probability only relies on the fact that the previous level verifies and thus by induction intermediate delegators can re-delegate to level L.

Definition 17 (DAC correctness). *A DAC scheme is correct if for all security parameters, λ, $L = O(\lambda)$, $L' \in [L]$, $L^* \in [L' - 1]$, $(\mathsf{pp}, td) \in \mathsf{Setup}(1^\lambda, 1^L)$, $(tsk, tpk) \in mathsfTKeyGen(\mathsf{pp})$, $(\mathsf{sk}_{rt}, \mathsf{pk}_{rt}) \in \mathsf{RootKeyGen}(\mathsf{pp})$, $(\mathsf{sk}_1, \mathsf{pk}_1) \in \mathsf{UKeyGen}(\mathsf{pp}, L')$, $(\mathsf{sk}_2, \mathsf{pk}_2) \in \mathsf{UKeyGen}(\mathsf{pp}, L')$ it holds that:*

Root issuance:

$$\Pr\left[\begin{pmatrix}\mathsf{Prove}(\mathsf{sk}_1, \mathsf{cred}_1, L') \\ \leftrightarrow \\ \mathsf{Verify}(\mathsf{pk}_{rt}, L', tpk)\end{pmatrix} = (1, *) \mid \begin{matrix}\mathsf{cred}_1 \in (\mathsf{Issue}(\mathsf{sk}_{rt}, \perp, \perp) \leftrightarrow \mathsf{Receive}(\mathsf{sk}_1, tok, L')) \\ \wedge L' = 1\end{matrix}\right] = 1$$

Delegator issuance:

$$\Pr\left[\begin{pmatrix}\mathsf{Prove}(\mathsf{sk}_2, \mathsf{cred}_2, L^*) \\ \leftrightarrow \\ \mathsf{Verify}(\mathsf{pk}_{rt}, L^*, tpk)\end{pmatrix} = (1, *) \mid \begin{matrix}\mathsf{cred}_2 \in (\mathsf{Issue}(\mathsf{sk}_1, \mathsf{cred}_1, L') \leftrightarrow \mathsf{Receive}(\mathsf{sk}_2, tok, L^*)) \\ \wedge(\mathsf{Prove}(\mathsf{sk}_1, \mathsf{cred}_1, L') \leftrightarrow \mathsf{Verify}(\mathsf{pk}_{rt}, L', tpk)) = (1, \_) \\ \wedge L^* = L' + 1\end{matrix}\right] = 1$$

Anonymity: Our anonymity definition is shown in Def 18. The anonymity game involves the adversary and the challenger. The adversary controls all participants, including the root credential authority (but does not control the TRA). The game proceeds as follows: The challenger generates the public parameters, which are given to the adversary along with the registrar's public key and ac-

cess to a registration and revocation oracle. The adversary creates two credential chains of the same length and provides the secret keys of the end users of these chains to the challenger. The challenger ensures that they are valid credential chains. The challenger randomly selects one of the users and proves possession of the corresponding credential chain to the adversary. The adversary wins if it can correctly identify which user the challenger picked. No honest users are created in this game, as the adversary controls all aspects except for the registration and revocation oracles. To model issuer privacy and showing privacy, the adversary outputs a bit, j, to indicate whether the challenger should act as the issuer or the shower. We formalize this game in Def. 18.

Definition 18 (Anonymity). *A DAC scheme is anonymous if the advantage any PPT adversary ($\mathcal{A} = \{\mathcal{A}_0, \mathcal{A}_1\}$) in the following anonymity game, defined by the chance that the game outputs* 1*, is* $1/2 + negl(\lambda)$*:*

1: $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^L)$
2: $(tsk, tpk) \leftarrow \mathsf{TKeyGen}(\mathsf{pp})$
3: $(st, j, \mathsf{pk}_{rt}, \mathsf{sk}_0, \mathsf{cred}_0, \mathsf{sk}_1, \mathsf{cred}_1, L') \leftarrow \mathcal{A}_0^{\mathcal{O}^{\mathsf{RegisterUser}}(\cdot), \mathcal{O}^{\mathsf{RevokeUser}}(\cdot)}(\mathsf{pp}, tpk)$
4: $\forall i \in \{0,1\}$:
5: **if** $(\mathsf{Prove}(\mathsf{pp}, \mathsf{sk}_i, \mathsf{cred}_i, L') \leftrightarrow \mathsf{Verify}(\mathsf{pk}_{rt}, L', tpk)) \neq (1, *)$, **return** $\perp$
6: $b \leftarrow\$ \{0,1\}$
7: **if** $j = 0, b' \leftarrow (\mathsf{Prove}(\mathsf{pp}, \mathsf{sk}_b, \mathsf{cred}_b, L') \leftrightarrow \mathcal{A}_1(st))$
8: **if** $j = 1, b' \leftarrow (\mathsf{Issue}(\mathsf{pp}, \mathsf{sk}_b, \mathsf{cred}_b, L' + 1) \leftrightarrow \mathcal{A}_1(st))$
9: **return** $b' = b$

Unlike the anonymity definition in [CL19], we allow the adversary to participate in the challenge credential chain. Therefore, we do not need to control the state of the game with the challenger; in the anonymity game, the challenger only performs the role of the TRA and the challenge user. In addition, we aim to maintain the anonymity of honest users even when the anonymity of some adversarial users is revoked. This new definition represents a simplified and more comprehensive anonymity model, which we present as a novel contribution.

Unforgeability: Our unforgeability game is simpler than [CL19], even though it is conceptually similar. We remove oracles that reveal pseudonyms of honest users. Revealing pseudonyms alone has no real-world use-case in DAC and the adversary effectively reveals pseudonyms during a showing anyway. Also, we integrate user creation with credential issuance, as a user's key pair is not used until it is associated with a credential. Otherwise, our unforgeability definition (Def. 19) is mostly unchanged from [CL19], but we add the RegisterUser and RevokeUser functions that facilitate revocation.

Moreover, our unforgeability definition ensures that the adversary was correctly delegated a credential on line 10 in Def. 19, and that none of the keys in the adversary's credential are on the deny list, on line 11.

To ensure that the challenger can check the key classes, we parameterize the definition with the extractor, $\mathcal{E}_{\mathsf{pk}}$, which takes in a public key and extracts a secret key from it. If $\mathcal{E}_{\mathsf{pk}}$ is run on the transcript of a showing, it extracts the secret key of the credential holder. If $\mathcal{E}_{\mathsf{pk}}$ is run on the transcript of an issuing, it extracts the secret key of the issuer. We denote these by $\mathcal{E}_{\mathsf{pk},R}$ for receiver and $\mathcal{E}_{\mathsf{pk},I}$ for issuer. This extractor must extract the same secret key no matter how the public key has been randomized. For mercurial signatures, this means that the extractor extracts a canonical secret key which is constant over any representation of the equivalence class of secret keys. We also assume an extractor $\mathcal{E}_{\mathsf{cred}}$ that can take in a credential or the transcript of a showing of a credential and output the canonical secret keys used in the delegation chain including the end user of the credential.

Definition 19 (Unforgeability). *A DAC scheme is unforgeable if any PPT adversary's advantage in the following game, defined by the chance that the game outputs* 1*, is negligible in* λ*. $\mathcal{A}$ is given all the oracles from Fig 6 labeled as $\mathcal{O}$.*

1: $(\mathsf{pp}, td) \leftarrow \mathsf{Setup}(1^\lambda, 1^L)$
2: $(tsk, tpk) \leftarrow \mathsf{TKeyGen}(\mathsf{pp})$
3: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{RootKeyGen}(\mathsf{pp})$
4: $\mathsf{SK}[0] = \mathsf{sk}; \mathsf{pk}_{rt} = \mathsf{pk}$
5: $(st, L) \leftarrow \mathcal{A}_0^{\mathcal{O}}(\mathsf{pp}, \mathsf{pk})$

6: $((b, *), \tau) \leftarrow (\mathsf{Verify}(\mathsf{pk}_{rt}, L, tpk) \leftrightarrow \mathcal{A}_1(st)$
7: $\{\mathsf{sk}_i\}_{i\in[L']} \leftarrow \mathcal{E}_{\mathsf{cred}}(\tau)$
8: **if** $forgery = 1$, **return** 1
9: **if** $\mathsf{sk}_0 \neq \mathsf{sk}_{rt}$, **return** b
10: **if** $\forall i \in [L'], (\mathsf{sk}_i, i) \notin \mathsf{DEL}_{\mathcal{A}}$, **return** b
11: **if** $\exists i \in [L']$, *s.t.* $\mathsf{sk}_i \in \mathsf{SK}_{DL}$, **return** b
12: **return** 0

4.3 DAC Construction

Our DAC construction uses a function $\mathsf{MultiSetup}$ which builds L sets of public parameters for our underlying mercurial signature scheme with $\ell = 2$ such that the schemes have the structure described in Sec. 3.3. We note that for the root authority we use the CL19 scheme [CL19] instead of our scheme as this is sufficient for the root. This function is described in Appendix A.

To simplify our DAC construction, we add a function $\mathsf{TracePK}$, which takes in a "linker" and a revocation token and returns if this linker is associated with the revocation token. These linker values will be stored in the deny list in the TRA's public key tpk.

Definition 20 (DAC construction).

- $\mathsf{Setup}(1^\lambda, 1^L) \rightarrow (\mathsf{pp}, td)$: Call the setup function described in Appendix A which generates L correlated parameters for our signature scheme in Fig. 3, $\{\mathsf{pp}_i\}_{i\in[L]} = \mathsf{MultiSetup}(1^\lambda, 1^{\ell=2}, 1^L)$. Then initialize extra bases for the revocation authority and the root authority using the CL19 scheme, $(\mathsf{pp}_{\mathsf{CL19}}) \leftarrow \mathsf{Setup}_{\mathsf{CL19}}(1^\lambda, 1^{\ell=2})$. Output $\mathsf{pp} = (\{\mathsf{pp}_i\}_{i\in[L]}, \mathsf{pp}_{\mathsf{CL19}}), td = (\{td_i\}_{i\in[\ell]})$.
- $\mathsf{RootKeyGen}(\mathsf{pp}) \rightarrow (\mathsf{sk}_{rt}, \mathsf{pk}_{rt})$: Generate a CL19 key pair using $\mathsf{pp}_{\mathsf{CL19}}$.
- $\mathsf{UKeyGen}(\mathsf{pp}, L') \rightarrow (\mathsf{sk}, \mathsf{pk})$: Create a secret key for the corresponding scheme, $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(\mathsf{pp}_{L'})$. The user initializes their credential chain as $chain = \perp$.
- $\mathsf{TKeyGen}(\mathsf{pp}) \rightarrow (tsk, tpk)$: Create a CL19 key $(\mathsf{sk}, \mathsf{pk})$ of length $\ell = 2$, $(tsk_{\mathsf{sk}}, tpk_{\mathsf{pk}}) \leftarrow \mathsf{KGen}(\mathsf{pp}_{\mathsf{CL19}})$. Initialize a set of linkers, $tsk_{link} = \emptyset$. Initialize a deny list, $DL = \emptyset$. Let $tsk_{\mathsf{sk}} = \mathsf{sk}$, $tsk = (tsk_{\mathsf{sk}}, tsk_{link})$, and $tpk = (tpk_{\mathsf{pk}}, DL)$.
- $\mathsf{RegisterUser}(\mathsf{pp}, tsk, \mathsf{pk}) \rightarrow (tok)$: Generate a new key pair $(\mathsf{sk}_{rev}, \mathsf{pk}_{rev})$ using $\mathsf{pp}_{\mathsf{CL19}}$. Sign pk_{rev} using tsk_{sk} where $tsk = (tsk_{\mathsf{sk}}, tsk_{link})$ yielding σ_0. Then, use sk_{rev} to sign pk, yielding σ_1. This yields the revocation token, $tok = (\mathsf{pk}_{rev}, \sigma_0, \sigma_1)$. The secret key, sk_{rev}, will serve as the linker for this revocation token and it is denoted as $link$. Save this linker in the TRA's state, $tsk'_{link} = tsk_{link} \cup \{link\}$ and update the state: $tsk' = (tsk_{\mathsf{sk}}, tsk'_{link})$. Output revocation token tok.
- $\mathsf{RevokeUser}(\mathsf{pp}, tsk, tpk, tok) \rightarrow tpk'$: Iterate through the linkers $(link_i)$ in tsk_{link} and check if $\mathsf{TracePK}(\mathsf{pp}, link_i, tok) = 1$ for each of them. If this holds for a linker, $link_i$, concatenate $link_i$ to the linkers in tpk (the deny list) and output this new public key as tpk'.
- $\mathsf{TracePK}(\mathsf{pp}, link, tok) \rightarrow \{0, 1\}$: Parse tok as $tok = (\mathsf{pk}_{rev}, \sigma_0, \sigma_1)$. Check if $\mathsf{RecognizePK}(\mathsf{pp}_{\mathsf{CL19}}, link, \mathsf{pk}_{rev}) = 1$ (cf. Sec. 2), i.e., parse pk_{rev} as $\mathsf{pk}_{rev} = (\hat{X}_1, \hat{X}_2)$. Parse $link = (x_1, x_2)$. Check if $\hat{X}_1^{x_2/x_1} = \hat{X}_2$. If this holds, output 1. Otherwise, output 0.
- $(\mathsf{Issue}(\mathsf{sk}_I, \mathsf{cred}_I, L') \leftrightarrow \mathsf{Receive}(\mathsf{sk}_R, tok, L')) \rightarrow \mathsf{cred}_R$: The receiver samples $\rho \leftarrow \mathcal{KC}$ (from the set of key converters) and generates a randomized public key from their secret key, pk', using the randomization factor, ρ. They also randomize their revocation token, tok, yielding, $tok = (\mathsf{pk}'_{rev}, \sigma'_0, \sigma'_1)$, such that $\mathsf{Verify}_{\mathsf{CL19}}(\mathsf{pp}_{\mathsf{CL19}}, tpk, \mathsf{pk}'_{rev}, \sigma'_0) = 1$ and $\mathsf{Verify}_{\mathsf{CL19}}(\mathsf{pp}_{\mathsf{CL19}}, \mathsf{pk}'_{rev}, \mathsf{pk}', \sigma'_0) = 1$. The receiver sends over pk'. The issuer then randomizes all public keys in their credential chain along with the signatures, randomizing their secret key to match. They also randomize all revocation tokens in their chain as described above. They then sign pk' yielding a signature, σ. They send their randomized credential chain, $chain$, along with σ to the receiver. The receiver computes the chain, $chain' = chain \| (\mathsf{pk}', \sigma, tok')$. The receiver stores their credential as $\mathsf{cred} = (chain', \rho)$. The randomizer is also stored to ensure the receiver can correctly randomize their secret key to match their public key in the chain.

– $(\mathsf{Prove}(\mathsf{sk}_P, \mathsf{cred}_P, L') \leftrightarrow \mathsf{Verify}(\mathsf{pk}_{rt}, L', tpk)) \rightarrow (b, \{tok_i\}_{i\in[L']})$: The prover randomizes all public keys and signatures in their credential cred using $\rho^* = \rho * \rho'$ where ρ is the randomizer found in their credential and ρ' is randomly sampled. They send over their randomized credential chain, *chain*, and perform an interactive proof of knowledge that they know the sk that corresponds to the last public key in the chain. The verifier then verifies each public key with the signatures. The verifier also iterates through the revocation tokens in the credential chain checks that for each public key pk_i and $tok_i = (\mathsf{pk}_{rev,i}, \sigma_{i,0}, \sigma_{i,1})$ in the chain it holds that $\mathsf{Verify}_{\mathsf{CL19}}(tpk, \mathsf{pk}_{rev,i}, \sigma_{i,0}) = 1$ and $\mathsf{Verify}_{\mathsf{CL19}}(\mathsf{pk}_{rev,i}, \mathsf{pk}_i, \sigma_{i,1}) = 1$. They then also iterate through each $link_j \in tpk$ and ensure that $\mathsf{TracePK}(\mathsf{pp}, link_j, tok_i) = 0$ for each level i in the length of the chain. If all these checks hold, the verifier outputs 1 and if any checks fail, the verifier outputs 0. The verifier also outputs all of the tok_i values received from the prover.

Theorem 21 (Correctness of the construction in Def. 20). *Our construction in Def. 20 is correct as defined in Def. 17.*

Theorem 22 (Unforgeability of the construction in Def. 20). *If the underlying signature scheme is unforgeable with respect to Def. 2, our construction in Def. 20 is unforgeable with respect to Def. 19.*

Theorem 23 (Anonymity of the construction in Def. 20). *If the underlying signature scheme has origin-hiding and has adversarial public key-class hiding, the DAC scheme in Def. 20 is anonymous with respect to definition 18.*

We prove these theorems in Appendix B.3.

5 Conclusion and Future Work

In this paper, we constructed mercurial signatures with stronger security properties than seen in the literature, which could potentially be used for many privacy-preserving schemes just as the first such signatures in [CL19]. We use it as a basis for an efficient DAC scheme with strong privacy guarantees and delegator revocation functionality. Our DAC construction could be further adapted to support attributes extending its functionality, where the technique from [PM24] seems promising. We leave this extension to future work.

6 Acknowledgments

We are very grateful to the anonymous reviewers for their many helpful comments and suggestions. Omid Mir was supported by the European Union's Horizon Europe project SUNRISE (project no. 101073821), and by PREPARED, a project funded by the Austrian security research programme KIRAS of the Federal Ministry of Finance (BMF). Scott Griffy and Anna Lysysanskaya were supported by NSF grants 2247305, 2154170, and 2312241 as well as the Ethereum Foundation.

References

AN11. Tolga Acar and Lan Nguyen. Revocation for delegatable anonymous credentials. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 423–440. Springer, Heidelberg, March 2011.

BB18. Johannes Blömer and Jan Bobolz. Delegatable attribute-based anonymous credentials from dynamically malleable signatures. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 18*, volume 10892 of *LNCS*, pages 221–239. Springer, Heidelberg, July 2018.

BCC+09. Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 108–125. Springer, Heidelberg, August 2009.

BCG+15. Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *2015 IEEE Symposium on Security and Privacy*, pages 287–304. IEEE Computer Society Press, May 2015.

BDG+23. Joakim Brorsson, Bernardo David, Lorenzo Gentile, Elena Pagnin, and Paul Stankovski Wagner. PAPR: Publicly auditable privacy revocation for anonymous credentials. In Mike Rosulek, editor, *CT-RSA 2023*, volume 13871 of *LNCS*, pages 163–190. Springer, Heidelberg, April 2023.

BGG19. Sean Bowe, Ariel Gabizon, and Matthew D. Green. A multi-party protocol for constructing the public parameters of the pinocchio zk-SNARK. In Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala, editors, *FC 2018 Workshops*, volume 10958 of *LNCS*, pages 64–77. Springer, Heidelberg, March 2019.

BS04. Dan Boneh and Hovav Shacham. Group signatures with verifier-local revocation. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick McDaniel, editors, *ACM CCS 2004*, pages 168–177. ACM Press, October 2004.

CDD17. Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. Practical UC-secure delegatable credentials with attributes and their application to blockchain. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 683–699. ACM Press, October / November 2017.

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

CKL+16. Jan Camenisch, Stephan Krenn, Anja Lehmann, Gert Læssøe Mikkelsen, Gregory Neven, and Michael Østergaard Pedersen. Formal treatment of privacy-enhancing credential systems. In Orr Dunkelman and Liam Keliher, editors, *SAC 2015*, volume 9566 of *LNCS*, pages 3–24. Springer, Heidelberg, August 2016.

CKS09. Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 481–500. Springer, Heidelberg, March 2009.

CKS10. Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. Solving revocation with efficient update of anonymous credentials. In Juan A. Garay and Roberto De Prisco, editors, *SCN 10*, volume 6280 of *LNCS*, pages 454–471. Springer, Heidelberg, September 2010.

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

CL02. Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 61–76. Springer, Heidelberg, August 2002.

CL06. Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 78–96. Springer, Heidelberg, August 2006.

CL19. Elizabeth C. Crites and Anna Lysyanskaya. Delegatable anonymous credentials from mercurial signatures. In Mitsuru Matsui, editor, *CT-RSA 2019*, volume 11405 of *LNCS*, pages 535–555. Springer, Heidelberg, March 2019.

CL21. Elizabeth C. Crites and Anna Lysyanskaya. Mercurial signatures for variable-length messages. *PoPETs*, 2021(4):441–463, October 2021.

CLPK22. Aisling Connolly, Pascal Lafourcade, and Octavio Perez-Kempner. Improved constructions of anonymous credentials from structure-preserving signatures on equivalence classes. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part I*, volume 13177 of *LNCS*, pages 409–438. Springer, Heidelberg, March 2022.

CS97. Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. *Technical Report/ETH Zurich, Department of Computer Science*, 260, 1997.

DHS15. David Derler, Christian Hanser, and Daniel Slamanig. A new approach to efficient revocable attribute-based anonymous credentials. In Jens Groth, editor, *15th IMA International Conference on Cryptography and Coding*, volume 9496 of *LNCS*, pages 57–74. Springer, Heidelberg, December 2015.

FHS19. Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. *Journal of Cryptology*, 32(2):498–546, April 2019.

GKM+18. Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-SNARKs. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 698–728. Springer, Heidelberg, August 2018.

GL24. Scott Griffy and Anna Lysyanskaya. PACIFIC. *IACR Communications in Cryptology*, 1(2), 2024.

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

HS14. Christian Hanser and Daniel Slamanig. Structure-preserving signatures on equivalence classes and their application to anonymous credentials. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 491–511. Springer, Heidelberg, December 2014.

HS21. Lucjan Hanzlik and Daniel Slamanig. With a little help from my friends: Constructing practical anonymous credentials. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2004–2023. ACM Press, November 2021.

MBG+23. Omid Mir, Balthazar Bauer, Scott Griffy, Anna Lysyanskaya, and Daniel Slamanig. Aggregate signatures with versatile randomization and issuer-hiding multi-authority anonymous credentials. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 30–44. ACM Press, November 2023.

MSBM23. Omid Mir, Daniel Slamanig, Balthazar Bauer, and René Mayrhofer. Practical delegatable anonymous credentials from equivalence class signatures. *Proc. Priv. Enhancing Technol.*, 2023(3):488–513, 2023.

NKTA24. Masaya Nanri, Octavio Perez Kempner, Mehdi Tibouchi, and Masayuki Abe. Interactive threshold mercurial signatures and applications. Cryptology ePrint Archive, Paper 2024/625, 2024. `https://eprint.iacr.org/2024/625`.

NRBB24. Valeria Nikolaenko, Sam Ragsdale, Joseph Bonneau, and Dan Boneh. Powers-of-tau to the people: Decentralizing setup ceremonies. In Christina Pöpper and Lejla Batina, editors, *Applied Cryptography and Network Security - 22nd International Conference, ACNS 2024, Abu Dhabi, United Arab Emirates, March 5-8, 2024, Proceedings, Part III*, volume 14585 of *Lecture Notes in Computer Science*, pages 105–134. Springer, 2024.

PM23. Colin Putman and Keith M. Martin. Selective delegation of attributes in mercurial signature credentials. In Elizabeth A. Quaglia, editor, *Cryptography and Coding - 19th IMA International Conference, IMACC 2023, London, UK, December 12-14, 2023, Proceedings*, volume 14421 of *Lecture Notes in Computer Science*, pages 181–196. Springer, 2023.

PM24. Colin Putman and Keith M. Martin. Selective delegation of attributes in mercurial signature credentials. In Elizabeth A. Quaglia, editor, *Cryptography and Coding*, pages 181–196, Cham, 2024. Springer Nature Switzerland.

Sho97. Victor Shoup. Lower bounds for discrete logarithms and related problems. In *Proceedings of the 16th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'97, page 256–266, Berlin, Heidelberg, 1997. Springer-Verlag.

Appendix

A Extendable Mercurial Signatures with an Updatable CRS

A.1 Updating the CRS

While we rely on a trusted setup (i.e. our construction is secure in the common reference string model) we describe a process to update the CRS which is very efficient. As described in [GKM+18], an updatable CRS has the functions: Setup, which generates the CRS, Update which updates the CRS and produces a proof of consistency, and $\mathsf{VerifyCRS}$ which takes in a proof and ensures that the CRS is correctly updated. Using these functions ensures that an efficient sequential setup can be performed and if only one of the users who include a proof in the setup is honest, the resulting CRS is secure (i.e. no party knows a full trapdoor that would allow them to break any security properties of the scheme). By "sequential" we mean that the parties that update the CRS do not have to be online at the same time, any updater can take an existing CRS and add their own trapdoors to it. We define these functions for our scheme below:

- $(\mathsf{pp}, \pi) \leftarrow \mathsf{Setup}(1^\lambda)$: Generate the public parameters as normal:
- $\mathsf{Setup}(1^\lambda, 1^\ell) \rightarrow (\mathsf{pp}, \pi)$:
  Sample $\{b_i, \hat{b}_i, \hat{d}_i, \hat{v}_i, v_i\}_{i\in[\ell]} \leftarrow\!\!\$\ \mathbb{Z}_p$.
  Compute:
  $\mathbf{B} = \{B_i\}_{i\in[2\ell]}$, where $\forall i \in [\ell], B_i \leftarrow P^{b_i}, B_{\ell+i} \leftarrow P^{b_i\hat{b}_i}$
  $\hat{\mathbf{B}} = \{\hat{B}_i\}_{i\in[2\ell]}$, where $\forall i \in [\ell], \hat{B}_i \leftarrow \hat{P}^{\hat{b}_i}, \hat{B}_{\ell+i} \leftarrow \hat{P}^{\hat{b}_i\hat{d}_i}$
  $\hat{\mathbf{V}} = \{\hat{V}_i\}_{i\in[2\ell]}$, where $\forall i \in [\ell], \hat{V}_i \leftarrow \hat{P}^{\hat{v}_i\hat{b}_i}, \quad \hat{V}_{\ell+i} \leftarrow \hat{P}^{\hat{v}_i}$
  $\mathbf{V} = \{V_i\}_{i\in[2\ell]}$, where $\forall i \in [\ell], V_i \leftarrow P^{v_i\hat{d}_i}, \quad V_{\ell+i} \leftarrow P^{v_i}$
  Output: $\mathsf{pp} = (\mathbf{B}, \hat{\mathbf{B}}, \hat{\mathbf{V}}, \mathbf{V})$. As the proof, π, commit to each of the trapdoors and produce a proof that each of the trapdoors is known (i.e. a proof of equality of discrete logarithm representation over the generators of the bilinear pairing). This can be done efficiently with Pedersen commitments

and Schnorr proofs [CS97] (in the ROM), using only 1 group element for the commitment and 2 elements in $\mathbb{Z}_p$ for each proof base with constant computation cost.

– $\mathsf{Update}(1^\lambda, \mathsf{pp}, (\pi_i)_{i\in[n]}) \to (\mathsf{pp}', \pi')$: Sample $\{b_i', \hat{b}_i, \hat{d}_i, \hat{v}_i, v_i\}_{i\in[\ell]} \leftarrow\$ \mathbb{Z}_p$ and compute:

$$\forall i \in [\ell], B_i' \leftarrow {B_i}^{b_i'},\ B_{\ell+i} \leftarrow {B_{\ell+i}}^{b_i'\hat{b}_i'}$$
$$\hat{B}_i' \leftarrow \hat{B}_i^{\hat{b}_i},\quad \hat{B}_{\ell+i}' \leftarrow \hat{B}_{\ell+i}^{\hat{b}_i'\hat{d}_i'}$$
$$\hat{V}_i' \leftarrow \hat{V}_i^{\hat{v}_i'\hat{d}_i'}, \hat{V}_{\ell+i} \leftarrow \hat{V}_{\ell+i}^{\hat{v}_i'}$$
$$V_i' \leftarrow {V_i}^{v_i'\hat{d}_i'}, {V_{\ell+i}}' \leftarrow ({V_{\ell+i}}')^{v_i'}$$

Output: $\mathsf{pp}' = (\mathbf{B}', \hat{\mathbf{B}}', \hat{\mathbf{V}}', \mathbf{V}')$. Commit to each trapdoor and prove that the resulting bases pp' were computed correctly over pp (using techniques similar to Setup). Output the Schnorr proofs and commitment as the proof, π. Include the old pp in the proof as well. This can be represented using Camenisch-Stadler notation: $\pi_R = NIZK[\{b_i, \hat{b}_i, \hat{d}_i, \hat{v}_i, v_i\}_{i\in[\ell]}, \{O_{b_i}, O_{\hat{b}_i}, O_{\hat{d}_i}, O_{\hat{v}_i}, O_{v_i}\}_{i\in[\ell]} : \forall i \in [\ell], C_{b_i} = \mathsf{Com}(b_i, O_{b_i}), C_{\hat{b}_i} = \mathsf{Com}(\hat{b}_i, O_{\hat{b}_i}), C_{\hat{d}_i} = \mathsf{Com}(\hat{d}_i, O_{\hat{d}_i}), C_{\hat{v}_i} = \mathsf{Com}(\hat{v}_i, O_{\hat{v}_i}), C_{v_i} = \mathsf{Com}(v_i, O_{v_i})\}_{i\in[\ell]}, B_i' = {B_i}^{b_i'},\ B_{\ell+i} = {B_{\ell+i}}^{b_i'\hat{b}_i'}, \hat{B}_i' = \hat{B}_i^{\hat{b}_i}, \hat{B}_{\ell+i}' = \hat{B}_{\ell+i}^{\hat{b}_i'\hat{d}_i'}, \hat{V}_i' = \hat{V}_i^{\hat{v}_i'\hat{d}_i'}, \hat{V}_{\ell+i} = \hat{V}_{\ell+i}^{\hat{v}_i'}, V_i' = {V_i}^{v_i'\hat{d}_i'}, {V_{\ell+i}}' = ({V_{\ell+i}}')^{v_i'}]$. Thus, π_R using Schnorr proof with Fiat-Shamir applied consists of the 5ℓ Pedersen commitments to the trapdoors, $\forall i \in [\ell], C_{b_i}, C_{\hat{b}_i}, C_{\hat{d}_i}, C_{\hat{v}_i}, C_{v_i}$, as well as 4ℓ commitments to the multiplications of those trapdoors, $\forall i \in [\ell], b_i\hat{b}_i, \hat{b}_i\hat{b}_i, \hat{v}_i\hat{b}_i, v_i\hat{d}_i$, which makes up 9ℓ group elements. The Schnorr proof itself then consists of 8ℓ first message "commitments" (which are group elements), for each base in the public parameters, as well as 9ℓ first message commitments to each of the commitments to the trapdoors (including the multiplications of trapdoors). We then need 18ℓ final messages for the Schnorr proof, one for each trapdoors, and one for each opening of a commitment to a trapdoor.

A.2 Extending the scheme to multiple levels

Extendability was an implicit property in [CL19] but was much simpler as the scheme was symmetric. Our scheme's parameters are structured in a way that we must have explicit functions to extend the scheme such that public keys at one level in the resulting DAC scheme can sign public keys from another level. To that end, we include algorithms, $\mathsf{ExtendSetup}$ and $\mathsf{FinalizeSetup}$, which are needed by the proof of our DAC scheme to ensure the scheme can be extended to multiple levels. We also update each of our security definitions to include an interactive setup which invokes these functions. The syntax of our extendable mercurial signature scheme is thus given by the functions in Def. 24. In Section Sec. A.3, we construct these functions which extend our scheme from Fig. 3.

To explain why we need such functions and properties for our proof, let's attempt to construct a reduction from APKCH to DAC anonymity where the APKCH challenger generates a single set of public parameters for one mercurial signature scheme, but the DAC scheme requires multiple levels of mercurial signature schemes such that public keys in the schemes can sign public keys from the next higher scheme. This reduction receives pp from the APKCH challenger. In the anonymity game, the reduction will need to make the credential depend on the APKCH challenger's secret bit. In our proof of anonymity, we'll create hybrids (shown in Def. 38) which replace each public key in the credential chain with a random public key (which, in our reduction is the APKCH challenger's public key). Thus, in our reduction, we need to replace one of the signatures in the credential chain with the signature which is returned by the APKCH challenger (σ^b in Fig. 2). To ensure that the adversary won't simply abort, we need this signature to appear as though it came from the DAC scheme. Thus, if this is some intermediate signature in the credential chain, σ^b must come from some public key, pk_i, be signed by some public key, pk_{i-1}, and sign another public key, pk_{i+1} to continue the chain. The natural public key for the reduction to use in place of pk_i is the pk^b returned by the PKCH challenger as this verifies with σ^b as this lets the reduction use the adversary's guess to guess b. But, if this signature is in the middle of a credential chain, we'll need to sign $\mathsf{pk}^b = \mathsf{pk}_i$ with some pk_{i-1}. But, pk^b is valid only for the public parameters that the APKCH challenger generated. Thus, this reduction must somehow include the pp generated by the APKCH challenger into the public parameters for

the DAC scheme. The public key bases for pp are $\hat{\mathbf{B}} = \{\hat{B}_i\}_{i\in[2\ell]}$ where $\forall i \in [\ell], \hat{B}_i = \hat{P}^{\hat{b}_i}, \hat{B}_{\ell+i} = \hat{P}^{\hat{b}_i\hat{d}_i}$. Let's assume the reduction has the secret key for the higher level scheme (whose public keys are in $\mathbb{G}_1$), $\mathsf{sk}_{i-1} = \{x_j\}_{j\in[\ell]}$. If our reduction computes the signature on pk^b in the normal way, $\sigma = (Z, Y, \hat{Y})$ where $Z = (\prod_{i\in[\ell]} \hat{X}_{\ell+i}^{x_i})^y, Y = P^{1/y}, \hat{Y} = \hat{P}^{1/y}$, and $\mathsf{pk}^b = \{\hat{X}_i\}_{i\in[\ell]}$, we can see that $e(Z, \hat{Y}) = \prod_{i\in[\ell]} \hat{P}^{\hat{b}_i\hat{d}_i\hat{x}_i x_i}$ where $\{\hat{x}_i\}_{i\in[\ell]}$ is the secret key for pk^b. Thus, to create public key bases for the signature scheme at level $i-1$, we would need to know $\hat{d}_i$ as we need to compute bases for the public key in $\mathbb{G}_1$ for pp_{i-1}. Unfortunately, the trapdoor, $\hat{d}_i$, is only known to the APKCH challenger, and thus, our reduction cannot generate these bases. In Section Sec. 3.3 we modified our scheme to output $P^{\hat{d}_i}$ (which is part of our solution for this problem), but then we run into a similar problem when we attempt to generate a pk_{i+1} (which is considered a message in the APKCH challenger's scheme) that verifies with the APKCH challenger's public parameters, pp, while still allowing the reduction to generate the public parameters for levels $i+1, \ldots, L$. This is because the scheme for pp_{i+1} must use the b_i trapdoors in its public key bases, but the APKCH challenger does not reveal $\hat{P}^{b_i}$ (which would allow the higher scheme to extend), but instead reveals P^{b_i}. Defining $\mathsf{ExtendSetup}$ and $\mathsf{FinalizeSetup}$ solves this problem and allows our proof of APKCH to focus on a single set of parameters.

Definition 24 (Extendable mercurial signatures).

- $\mathsf{Setup}(1^\lambda, 1^\ell) \to (\mathsf{pp}, td)$*: Outputs public parameters* pp*, including parameterized equivalence relations for the message, public key, and secret key spaces:* $\mathcal{R}_M$, $\mathcal{R}_{\mathsf{pk}}$, $\mathcal{R}_{\mathsf{sk}}$ *and the sample space for key and message converters. This function also outputs a trapdoor (*td*) that can be used (in conjunction with the corresponding secret key) to recognize public keys.*
- $\mathsf{ExtendSetup}(\mathsf{pp}') \to (\mathsf{pp}, td)$*: Extends a scheme (described by* pp'*) such that the outputted scheme defined by* pp *can be used to sign the public keys of the scheme defined by* pp^* *where* $\mathsf{pp}^* \leftarrow \mathsf{FinalizeSetup}(\mathsf{pp}', td, td')$ *where* td' *is the trapdoor for* pp'*.*
- $\mathsf{FinalizeSetup}(\mathsf{pp}, td, td') \to (\mathsf{pp}^*, td^*)$*: Finalize a previously generated scheme,* (pp, td)*, after it has been extended, where* td' *is the trapdoor from the extension.*
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{sk})$*: Generates a key pair.*
- $\mathsf{Sign}(\mathsf{sk}, M) \to \sigma$*: Signs a message* M *with the given secret key.*
- $\mathsf{Verify}(\mathsf{pk}, M, \sigma) \to (0$ *or* $1)$*: Returns 1 iff* σ *is a valid signature for* M *w.r.t.* pk*.*
- $\mathsf{ConvertPK}(\mathsf{pk}, \rho) \to \mathsf{pk}'$*: Given a key converter* ρ*, returns* pk' *by randomizing* pk *with* ρ*.*
- $\mathsf{ConvertSK}(\mathsf{sk}, \rho) \to \mathsf{sk}'$*: Randomize a secret key such that it now corresponds to a public key which has been randomized with the same* ρ *(i.e. signatures from* $\mathsf{sk}' = \mathsf{ConvertSK}(\mathsf{sk}, \rho)$ *verify by the randomized* $\mathsf{pk}' = \mathsf{ConvertPK}(\mathsf{pk}, \rho)$*).*
- $\mathsf{ConvertSig}(\mathsf{pk}, M, \sigma, \rho) \to \sigma'$*: Randomize the signature so that it verifies with a randomized* pk' *(which has been randomized with the same* ρ*) and* M*, but* σ' *is unlinkable to* σ*.*
- $\mathsf{ChangeRep}(\mathsf{pk}, M, \sigma, \mu) \to (M', \sigma')$*: Randomize the message-signature pair such that* $\mathsf{Verify}(\mathsf{pk}, M', \sigma') = 1$ *(i.e.,* σ' *and* σ *are indistinguishable) where* M' *is a new representation of the message equivalence class,* $[M]_{\mathcal{R}_M}$*.*
- $\mathsf{VerifyKey}(\mathsf{pp}, \mathsf{pk}) \to \{0, 1\}$*: Takes a public key and verifies if it is well-formed w.r.t public parameters* pp*.*
- $\mathsf{VerifyMsg}(\mathsf{pp}, M) \to \{0, 1\}$*: Takes a message and verifies if it is well-formed w.r.t public parameters* pp*.*

Definition 25 (Correctness). *A mercurial signature for parameterized equivalence relations,* $\mathcal{R}_{\mathcal{M}}$*,* $\mathcal{R}_{\mathsf{pk}}$*,* $\mathcal{R}_{\mathsf{sk}}$*, message randomizer space,* sample_μ*, and key randomizer space,* sample_ρ*, is correct if for all* (λ, ℓ)*,* $\forall(\mathsf{pp}, td) \in \mathsf{Setup}(1^\lambda, 1^\ell)$*, and* $\forall(\mathsf{sk}, \mathsf{pk}) \in \mathsf{KGen}(1^\lambda)$*, the following holds:*

- ***Verification.*** $\forall M \in \mathcal{M}, \sigma \in \mathsf{Sign}(\mathsf{sk}, M) : \mathsf{Verify}(\mathsf{pk}, M, \sigma) = 1 \wedge \mathsf{VerifyMsg}(\mathsf{pp}, M) = 1 \wedge \mathsf{VerifyKey}(\mathsf{pp}, \mathsf{pk}) = 1$.

- ***Key conversion.*** $\forall \rho \in \mathsf{sample}_\rho, (\mathsf{ConvertPK}(\mathsf{pk}, \rho), \mathsf{ConvertSK}(\mathsf{sk}, \rho)) \in \mathsf{KGen}(1^\lambda)$, $\mathsf{ConvertSK}(\mathsf{pk}, \rho) \in [\mathsf{sk}]_{\mathcal{R}_{\mathsf{sk}}}$, *and* $\mathsf{ConvertPK}(\mathsf{pk}, \rho) \in [\mathsf{pk}]_{\mathcal{R}_{\mathsf{pk}}}$.
- ***Signature conversion.*** $\forall M \in \mathcal{M}, \sigma, \rho \in \mathsf{sample}_\rho, \sigma', \mathsf{pk}'$ *s.t* $\mathsf{Verify}(\mathsf{pk}, M, \sigma) = 1$, $\sigma' = \mathsf{ConvertSig}(\mathsf{pk}, M, \sigma, \rho)$, *and* $\mathsf{pk}' = \mathsf{ConvertPK}(\mathsf{pk}, \rho)$, *then* $\mathsf{Verify}(\mathsf{pk}', M, \sigma') = 1$.
- ***Change of message representation.*** $\forall M \in \mathcal{M}, \sigma, \mu \in \mathsf{sample}_\mu, M', \sigma'$ *such that* $\mathsf{Verify}(\mathsf{pk}, M, \sigma) = 1$ *and* $(M', \sigma') = \mathsf{ChangeRep}(\mathsf{pk}, M, \sigma; \mu)$ *then* $\mathsf{Verify}(\mathsf{pk}, M', \sigma') = 1$ *and* $M' \in [M]_{\mathcal{R}_M}$.
- ***Extension.*** $\forall\ (\mathsf{pp}, td) \in \mathsf{Setup}(1^\lambda, 1^\ell)$, $(\mathsf{pp}', td') \in \mathsf{ExtendSetup}(\mathsf{pp})$, $(\mathsf{pp}^*, td^*) \in \mathsf{FinalizeSetup}(\mathsf{pp}, td, td')$, *it holds that* $(\mathsf{pp}^*, td^*) \in \mathsf{Setup}(1^\lambda, 1^\ell)$. *Further,* $\mathcal{M}' = \mathcal{PK}^*$ *where* $\mathcal{M}'$ *and* $\mathcal{PK}^*$ *are the message and public key space for* pp' *and* pp^*.

To ensure our scheme stays secure when extended, we define an interactive setup protocol with an adversary ($\mathsf{Setup}^{\mathsf{ext}}$ in Fig. 7) which either generates a scheme with Setup then interacts with the adversary to extend it, or extends the parameters that an adversary generates. This is used by the unforgeability and class-hiding properties in Definitions 2 and 8

$(\mathsf{Setup}^{\mathsf{ext}}(1^\lambda, 1^\ell) \leftrightarrow \mathcal{A}) \rightarrow (\mathsf{pp}, td)$
1: $\mathsf{pp}_0 \leftarrow \mathcal{A}(1^\lambda, 1^\ell)$
2: **if** $\mathsf{pp}_0 = \perp$,
3: $\quad(\mathsf{pp}_1, td_1) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$
4: $\quad td' \leftarrow \mathcal{A}(\mathsf{pp}_1)$
5: $\quad(\mathsf{pp}, td) \leftarrow \mathsf{FinalizeSetup}(\mathsf{pp}_1, td_1, td')$
6: **else** ,
7: $\quad(\mathsf{pp}_1, td_1) \leftarrow \mathsf{ExtendSetup}(\mathsf{pp}_0)$
8: $\quad$**return** (pp_1, td_1)

Fig. 7. An interactive setup extension function for security games

Definition 26 (Unforgeability under extension). *A mercurial signature scheme for parameterized equivalence relations* $\mathcal{R}_\mathcal{M}$, $\mathcal{R}_{\mathsf{pk}}$, $\mathcal{R}_{\mathsf{sk}}$, *is unforgeable if for all parameters* (λ, ℓ) *and all probabilistic, polynomial-time (PPT) algorithms,* $\mathcal{A}$, *having access to a signing oracle, there exists a negligible function negl such that:*

$$\Pr\left[\begin{array}{r|l} \mathsf{Verify}(pk^*, M^*, \sigma^*) = 1 & \mathsf{PP} \leftarrow (\mathsf{Setup}^{\mathsf{ext}}(1^\lambda, 1^\ell) \leftrightarrow \mathcal{A}); \\ \wedge [pk^*]_{\mathcal{R}_{\mathsf{pk}}} = [pk]_{\mathcal{R}_{\mathsf{pk}}} & (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}); \\ \wedge \forall M \in Q, [M^*]_{\mathcal{R}_M} \neq [M]_{\mathcal{R}_M} & (\mathsf{pk}^*, M^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(sk, \cdot)}(\mathsf{pk}) \end{array}\right] \leq negl(\lambda) \tag{1}$$

Where Q *is the list of messages that the adversary queried to the* Sign *oracle.*

Definition 27 (Adversarial public key class-hiding under extension). *A mercurial signature,* Γ, *has adversarial public key class-hiding if the advantage of any PPT set of algorithms* $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$, *the following advantage* $\mathsf{Adv}^{\mathsf{APKCH}}_{\Gamma,\mathcal{A}}(\lambda)$ *is negligible,*

$$\mathsf{Adv}^{\mathsf{APKCH}}_{\Gamma,\mathcal{A}}(\lambda) := \left|\Pr\left[\boldsymbol{Exp}^{\mathsf{APKCH},0}_{\Gamma,\mathcal{A}}(\lambda) = 1\right] - \Pr\left[\boldsymbol{Exp}^{\mathsf{APKCH},1}_{\Gamma,\mathcal{A}}(\lambda) = 1\right]\right|$$

where $\boldsymbol{Exp}^{\mathsf{APKCH},b}_{\Gamma,\mathcal{A}}(\lambda)$ *is the experiment shown in Figure 8.*

A.3 Construction of Extendable Mercurial Signatures

We now complete our construction from Sec. 3.3. As described in Appendix A.2, we need to define functions to extend and finalize parameters, i.e., we need to ensure that our signature scheme is secure when accepting a D_i from the higher level scheme. This requires extra work, as for the lower

$(\mathsf{pp} \leftarrow (\mathsf{Setup}^{\mathsf{ext}}(1^\lambda, 1^\ell) \leftrightarrow \mathcal{A});$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(\mathsf{pp});$
$(\mathsf{pk}_{\mathcal{A}}, \sigma_{\mathcal{A}}, M, \mathsf{st}) \leftarrow \mathcal{A}_1^{\mathcal{O}^{\mathsf{Sign}}(\mathsf{sk},\cdot)}(\mathsf{pk}, \mathsf{pp})$
$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, M)$
$\rho_0 \leftarrow\!\!\$\ \mathbb{Z}_p^*; \mathsf{pk}^0 \leftarrow \mathsf{ConvertPK}(\mathsf{pk}, \rho_0);$
$\sigma^0 \leftarrow \mathsf{ConvertSig}(\sigma, \rho_0)$
$\rho_1 \leftarrow\!\!\$\ \mathbb{Z}_p^*; \mathsf{pk}^1 \leftarrow \mathsf{ConvertPK}(\mathsf{pk}_{\mathcal{A}}, \rho_1);$
$\sigma^1 \leftarrow \mathsf{ConvertSig}(\sigma_{\mathcal{A}}, \rho_1)$
if $\mathsf{Verify}(\mathsf{pk}_{\mathcal{A}}, \sigma_{\mathcal{A}}, M) = 1 \wedge \mathsf{VerifyMsg}(\mathsf{pp}, M) = 1 \wedge \mathsf{VerifyKey}(\mathsf{pp}, \mathsf{pk}_{\mathcal{A}}) = 1$
  return $\mathcal{A}_2^{\mathcal{O}^{\mathsf{Sign}}(\mathsf{sk},\cdot)}(\mathsf{pp}, \mathsf{st}, \mathsf{pk}^b, \sigma^b)$
else return $\mathcal{A}_2^{\mathcal{O}^{\mathsf{Sign}}(\mathsf{sk},\cdot)}(\mathsf{pp}, \mathsf{st}, \perp, \perp)$

Fig. 8. Adversarial public key class-hiding under extension experiment $\mathbf{Exp}^{\mathsf{APKCH},b}_{\Gamma,\mathcal{A}}(\lambda)$.

level scheme to remain secure, we need to inject new unknown trapdoors into $\hat{\mathbf{B}}'$. We describe this process in the ExtendSetup function in Fig. 9. To ensure the higher scheme is then correct after this modification, we also have to update the $\hat{\mathbf{B}}$ vector in the higher level scheme using the trapdoor from the lower level scheme, which we describe in the FinalizeSetup function in Fig. 9. We use a Setup function similar to the one for $L = 1$ we described in Fig. 3 but which also outputs $D_i = P^{\hat{d}_i}$. This change explains why the pp' passed to ExtendSetup in Fig. 9 includes D_i.

We now describe the function, MultiSetup, in Def. 28 used to generate multiple correlated mercurial signature schemes. This function will output parameters identical to Fig. 4 but calls our ExtendSetup and FinalizeSetup functions. These functions can then be replaced with our challenger in Fig. 8 in order to prove it secure. We can see by correctness in Def. 1 that because we've called ExtendSetup and FinalizeSetup, with the correct values, by the **Extension** correctness property our resulting parameters will be in the possible output of $\mathsf{Setup}(1^\lambda, 1^\ell)$ and thus they satisfy the rest of the properties in the correctness definition and their message and public key spaces will be correctly linked to create delegation chains.

Definition 28 (Generating multiple levels of correlated mercurial signatures with APKCH).

- $\mathsf{MultiSetup}(1^\lambda, 1^2, 1^L) \to \{\mathsf{pp}_i\}_{i\in[L]}$: *Compute:* $(\mathsf{pp}'_L, td'_L) = \mathsf{Setup}(1^\lambda, 1^{\ell=2})$ *and then call* $\forall i \in [L], (\mathsf{pp}'_{i-1}, td'_i) \leftarrow \mathsf{ExtendSetup}(\mathsf{pp}'_i)$. *To compute* ExtendSetup *correctly, an implementation will work backwards from* $i = L$. *Next, call* $\forall i \in [L], (\mathsf{pp}_i, td_i) \leftarrow \mathsf{FinalizeSetup}(\mathsf{pp}'_i, td'_i, td'_{i-1})$ *to finish the parameters and trapdoors,* $\{\mathsf{pp}_i, td_i\}_{i\in[L]}$ *(the final level of parameters,* pp_0*, do not need to be finalized as they have not been extended). The mercurial signature schemes are extended as described in Sec. 3.3.*

Correctness. We can see that the ExtendSetup function will exponentiate the public key bases from the higher scheme, $\hat{B}'_i$ and $\hat{B}'_{\ell+i}$ with new trapdoors, b_i and $\hat{b}_i$. Thus, during FinalizeSetup, the setup must exponentiate with these trapdoors to "fix" the scheme, making the two schemes work together again.

How extension impacts our security proofs. After the scheme is extended, the higher scheme which this was extended from will use the bases for public keys attained from the lower scheme as long as they were generated honestly (we model this by having the lower parameter generation function output the discrete logs used to update the bases so that the higher scheme can inspect them). When proving our construction with extension secure in Appendix B, our reduction receives these trapdoors, $\hat{b}'_i, \hat{d}'_i, \hat{v}'_i$ from the adversary (either in ExtendSetup or FinalizeSetup) and because each of them is multiplied with a random scalar, the resulting $\hat{b}^*_i, \hat{d}^*_i, \hat{v}^*_i$ are effectively random, and thus our reduction can proceed as if it had generated the trapdoors itself. We're also able to keep the generation of D_i out of any of our generic group model proofs by first proving Lemma 36 without these D_i values and then reducing to this lemma in the standard model while generating D_i values while proving Thm. 12. We can see that while most of the parameters are effectively random in our reduction using the extension functions, our ExtendSetup function outputs exactly $\forall i \in [\ell], B_i = (\hat{B}'_i)$. This is not a problem since it just invalidates message class hiding. We only use message class hiding

– $\mathsf{ExtendSetup}(1^\lambda, \mathsf{pp}') \to (\mathsf{pp}, td)$:
Parse $\mathsf{pp}' = \{D_i, (B'_i), (B'_{\ell+i}), (\hat{B}'_i), (\hat{B}'_{\ell+i}), V'_i, \hat{V}'_i\}$.
Sample: $b'_i, \hat{b}_i, \hat{d}_i, v_i, \hat{v}_i \leftarrow\!\!\$\ \mathbb{Z}_p$

$\forall i \in [\ell]$,

$$B_i \leftarrow (\hat{B}'_i), \quad B_{\ell+i} \leftarrow (\hat{B}'_{\ell+i})^{\hat{b}_i}$$
$$\hat{B}_i \leftarrow (D'_i)^{\hat{b}_i}, \quad \hat{B}_{\ell+i} \leftarrow (D'_i)^{\hat{b}_i \hat{d}_i}$$
$$\hat{V}_i \leftarrow P^{\hat{v}_i \hat{d}_i}, \quad \hat{V}_{\ell+i} \leftarrow P^{\hat{v}_i}$$
$$V_i \leftarrow (\hat{V}'_i)^{v_i \hat{d}_i}, V_{\ell+i} \leftarrow (\hat{V}'_{\ell+i})^{v_i}$$

$\mathbf{B^l} = \{B_i\}_{i\in[\ell]}, \mathbf{B^u} = \{B_{\ell+i}\}_{i\in[\ell]}$,
$\hat{\mathbf{B}}^\mathbf{l} = \{\hat{B}_i\}_{i\in[\ell]}, \hat{\mathbf{B}}^\mathbf{u} = \{\hat{B}_{\ell+i}\}_{i\in[\ell]}$,
$\hat{\mathbf{V}}^\mathbf{l} = \{\hat{V}_i\}_{i\in[\ell]}, \hat{\mathbf{V}}^\mathbf{u} = \{\hat{V}_{\ell+i}\}_{i\in[\ell]}$,
$\mathbf{V^l} = \{V_i\}_{i\in[\ell]}, \mathbf{V^u} = \{V_{\ell+i}\}_{i\in[\ell]}$,
Output: $\mathsf{pp} = (\mathbf{B^l}, \mathbf{B^u}, \hat{\mathbf{B}}^\mathbf{l}, \hat{\mathbf{B}}^\mathbf{u}, \hat{\mathbf{V}}^\mathbf{l}, \hat{\mathbf{V}}^\mathbf{u}, \mathbf{V^l}, \mathbf{V^u})$
and $td = \{\hat{b}_i, \hat{d}_i, v_i, \hat{v}_i\}_{i\in[\ell]}$

– $\mathsf{FinalizeSetup}_i(1^\lambda, \mathsf{pp}, td, td') \to (\mathsf{pp}^*)$:
Parse $\mathsf{pp} = \{D_i, B_i, B_{\ell+i}, \hat{B}_i, \hat{B}_{\ell+i}, V_i, \hat{V}_i\}_{i\in[\ell]}$.
Parse $td = \{\hat{b}_i, \hat{d}_i, v_i, \hat{v}_i\}_{i\in[\ell]}$.
Parse $td' = \{b'_i, \hat{b}'_i, \hat{d}'_i, v'_i, \hat{v}'_i\}_{i\in[\ell]}$.
Compute $\mathsf{pk}^* = \{\mathbf{B^{l*}}, \mathbf{B^{u*}}, \hat{\mathbf{B}}^{\mathbf{l}*}, \hat{\mathbf{B}}^{\mathbf{u}*}, \mathbf{V^{l*}}, \mathbf{V^{u*}}, \hat{\mathbf{V}}^{\mathbf{l}*}, \hat{\mathbf{V}}^{\mathbf{u}*}\}$ where:
$\mathbf{B^{l*}} = \{B_i\}_{i\in[\ell]}$
$\mathbf{B^{u*}} = \{B_{\ell+i}\}_{i\in[\ell]}$
$\hat{\mathbf{B}}^{\mathbf{l}*} = \{\hat{B}_i\}_{i\in[\ell]}$
$\hat{\mathbf{B}}^{\mathbf{u}*} = \{(\hat{B}_{\ell+i})^{\hat{b}'_i}\}_{i\in[\ell]}$
$\mathbf{V^{l*}} = \{(V_i)^{\hat{b}'_i}\}_{i\in[\ell]}$
$\mathbf{V^{u*}} = \{(V_{\ell+i})\}_{i\in[\ell]}$
$\hat{\mathbf{V}}^{\mathbf{l}*} = \{(\hat{V}_i)\}_{i\in[\ell]}$
$\hat{\mathbf{V}}^{\mathbf{u}*} = \{(\hat{V}_{\ell+i})\}_{i\in[\ell]}$
Output pp^*

Fig. 9. Setup that can sign public keys from another scheme.

for the highest level in our DAC scheme and thus, we do not need if for the intermediate levels (which are generated from the $\mathsf{ExtendSetup}$).

B Proofs

We first prove (in Sec. B.1) properties where the public parameters generated in the games are simply outputted by Setup instead of being outputted by $\mathsf{Setup}^{\mathsf{ext}}$ as described in Fig. 7 in Sec. A. We then prove that our properties still hold when $\mathsf{Setup}^{\mathsf{ext}}$ is used in Sec. B.2.

B.1 Proofs for $L = 1$

Correctness of the mercurial signature scheme in Fig. 3. Verification. We can see that if $M \in \mathcal{M}$, then $\mathsf{VerifyMsg}(\mathsf{pp}, M) = 1$, and the message has the structure, $M = ({B_1}^{m_1}, {B_2}^{m_2}, ..., {B_\ell}^{m_\ell}, {B_{\ell+1}}^{m_1}, ..., {B_{\ell+\ell}}^{m_\ell}) = (P^{b_1 m_1}, P^{b_2 m_2}, ..., P^{b_\ell m_\ell}, P^{b_1 \hat{b}_1 m_1}, ..., P^{b_\ell \hat{b}_\ell m_\ell})$, then when the message is signed with $\mathsf{sk} = (x_1, .., x_\ell)$, then it will verify with the lower half of pk: $\mathsf{pk}^l = (X_1, ..., X_\ell) = (P^{\hat{b}_1 m_1}, P^{\hat{b}_2 m_2}, ..., P^{\hat{b}_\ell m_\ell})$ i.e.: $e(Z, Y) = e(\prod_{i\in[\ell]} P^{b_i b_i m_i x_i}, P) = \prod_{i\in[\ell]} e(M_i, \hat{X}_i)$. **Key conversion.** We can see that because keys are defined by the discrete log between elements, when we raise them all to the same power, they are in the same equivalence class. **Signature conversion.** We can see that

raising Z by ϕ as well as $Y, \hat{Y}$ by $1/\phi$, this effectively translates the y value but the signature remains correct. Raising Z by ρ effectively makes it as though it were signed by $\rho\mathsf{sk}$ instead of the original sk as when we look at the discrete log of Z, $\prod_{i\in[\ell]} P^{b_i b_i m_i x_i}$ the exponent is a sum of monomials, each of which contains exactly one distinct part of the secret key, x_i. Thus, raising Z to the ρ power multiplies each of these, exactly how $\rho\mathsf{sk}$ operates. **Change of message representation.** Similar to our proof of correct signature conversion, raising by ϕ effectively change the y used for the signature and we see the discrete log of Z is a sum of monomials with each having exactly one distinct element of the message, m_i. Thus, raising Z to the μ power multiplies each m_i making it verify with M^μ.

Unforgeability of the mercurial signature scheme in Fig. 3

Proof of Thm. 11 (Unforgeability of the construction in Fig. 3). This proof reduces the unforgeability of our scheme to that of the original mercurial signature construction in [CL19]. To prove this, we give a reduction that, given a forger for our scheme, outputs a forgery for the Crites-Lysyanskaya mercurial signature scheme. Intuitively, this reduction works for two reasons: (1) the signature and verification algorithms of our scheme are identical to that of [CL19] (ignoring verification, which makes our construction stricter on what is accepted) and (2) the public parameters will be generated by the reduction, and thus, the reduction will be able to transform public keys and signatures from either scheme so that they appear as though they were generated from the other scheme.

Our reduction takes as input a public key $\mathsf{pk} = \{\hat{X}_1, \ldots, \hat{X}_\ell\}$ from its unforgeability challenger and needs to form a public key pk' to forward to the adversary for our scheme. It forms $\mathsf{pk}' = \{\hat{X}_1^{\hat{b}_1}, \hat{X}_2^{\hat{b}_2}, ..., \hat{X}_\ell^{\hat{b}_\ell}, \hat{X}_1^{\hat{d}_1\hat{b}_1}, \hat{X}_2^{\hat{d}_2\hat{b}_2}, ..., \hat{X}_\ell^{\hat{d}_\ell\hat{b}_\ell}\}$ where $\mathsf{pk} = \{\hat{X}_1, ..., \hat{X}_\ell\}$ is the public key from the challenger.

The reduction is now ready to receive signature queries from the adversary, and respond to them. When the adversary queries the reduction on message $\mathbf{M}$, the first step is to verify that the messages is in the message space using $\mathsf{VerifyMsg}(\mathsf{pp}, \mathbf{M})$. Next, the reduction will parse the message $\mathbf{M} = (M_1, \ldots, M_{2\ell})$ into the lower half $\mathbf{M}^l = (M_1, \ldots, M_\ell)$ and upper half $\mathbf{M}^u = (M_{\ell+1}, \ldots, M_{2\ell})$. It queries the Crites-Lysyanskaya mercurial signature challenger on the message $\mathbf{M}^u$ and receives the signature $\sigma = (Z, Y, \hat{Y})$, and returns it to the adversary. We can see by inspection of the this signature computed the same way as our signature for the public key pk'.

Finally, the adversary outputs its forgery: a message $\mathbf{M}^*$ and a signature σ^* under public key pk^*. For the forged message, $\mathbf{M}^*$, the reduction will use the upper half of the message vector output by the adversary, the lower half of the public key vector exponentiated with the inverse of the trapdoor $(\hat{b}_i)$, and will output the adversary's signature unchanged. Because the signature verification of our scheme is identical to that of [CL19], this is guaranteed to verify under [CL19] without changing the σ^* outputted by the adversary. To formalize this proof, we show the reduction in Fig. B.1 and analyze it.

Analysis. We know that the adversary's outputted message, public key, and signature will all verify. Thus, we have that $e(Z^{\mathcal{A}}, Y^{\mathcal{A}}) = e(M_i^{\mathcal{A}}, \hat{X}_i^{\mathcal{A}})$, $e(M_i^{\mathcal{A}}, \hat{V}_i) = e(M_{\ell+i}^{\mathcal{A}}, \hat{V}_{\ell+i})$, and $e(V_i, \hat{X}_i^{\mathcal{A}}) = e(V_{\ell+i}, \hat{X}_{\ell+i}^{\mathcal{A}})$. We also know that $\mathsf{pk}^{\mathcal{A}}$ is in the same equivalence class as pk' and thus, $\exists\rho$ such that $(\mathsf{pk}')^\rho = \mathsf{pk}^{\mathcal{A}}$. Because $e(M_i^{\mathcal{A}}, \hat{V}_i) = e(M_{\ell+i}^{\mathcal{A}}, \hat{V}_{\ell+i})$, we know that $M_{\ell+i} = M_i^{\hat{b}_i}$. Thus, we can see that the modified public key, pk^* verifies with $\mathbf{M}^*$: $e(M_{\ell+i}, (\hat{X}_i^{\mathcal{A}})^{1/\hat{b}_i}) = e(M_{\ell+i}^{1/\hat{b}_i}, (\hat{X}_i^{\mathcal{A}})) = e(M_i, (\hat{X}_i^{\mathcal{A}})) = e(Z, \hat{Y})$. We can see that because the lower half of pk' is simply the challenge public key from CL19 raised pair-wise to the $\hat{b}_i$ power, then because pk^* is raised to the inverse of $\hat{b}_i$ power, then if $\mathsf{pk}^{\mathcal{A}}$ is in the same equivalence class as pk', then pk^* is in the same equivalence class as pk. Thus, our reduction's outputted forgery, $\mathbf{M}^*, \mathsf{pk}^*, \sigma^{\mathcal{A}}$, is valid and pk^* is in the same equivalence class as pk. Furthermore, if $\mathbf{M}^{\mathcal{A}}$ doesn't belong to the equivalence class of any message submitted to the reduction's signature oracle by the adversary, it's also not in the equivalence class of any message submitted to the CL19 signature oracle, making it a valid forgery.

Proof of Thm. 12 (APKCH of the construction in Fig. 3). We show an overview of the proof structure in Fig. 11. We will first prove that we can extract the discrete logs of messages and

$\mathcal{R}^{\mathcal{A},\mathcal{O}^{\mathsf{Sign},\mathsf{CL19}}}(\mathsf{pp}_{\mathsf{CL19}},\mathsf{pk}) \to (\pi^*,\mathbf{M}^*,\sigma^*)$:

1: $\mathbf{b} = \{b_i\}_{i\in[\ell]} \leftarrow\$ \mathbb{Z}_p$
2: $\hat{\mathbf{b}} = \{\hat{b}_i\}_{i\in[\ell]} \leftarrow\$ \mathbb{Z}_p$
3: $\hat{\mathbf{d}} = \{\hat{d}_i\}_{i\in[\ell]} \leftarrow\$ \mathbb{Z}_p$
4: $\hat{\mathbf{v}} = \{\hat{v}_i\}_{i\in[\ell]} \leftarrow\$ \mathbb{Z}_p$
5: $\mathbf{v} = \{v_i\}_{i\in[\ell]} \leftarrow\$ \mathbb{Z}_p$
6: $\mathsf{pp} = \{D_i, B_i, B_{\ell+i}, \hat{B}_i, \hat{B}_{\ell+i}, V_i, V_{i+\ell}, \hat{V}_i, \hat{V}_{i+\ell}\}_{i\in[\ell]}$ where $\forall i \in [\ell], B_i = P^{b_i}$, $B_{\ell+i} = P^{b_i\hat{b}_i}$, $\hat{B}_i = \hat{P}^{\hat{b}_i\hat{d}_i}$, $D_i = P^{\hat{d}_i}$, $\hat{B}_{\ell+i} = \hat{B}_i^{\hat{d}_i} =$, $V_i = P^{v_i\hat{d}_i}$, $V_{i+\ell} = P^{v_i\hat{d}_i}$, $\hat{V}_i = \hat{B}_i^{v_i}$, and $\hat{V}_{i+\ell} = P^{v_i}$
7: Parse $\mathsf{pk} = \{\hat{X}_1, \ldots, \hat{X}_\ell\}$
8: $\mathsf{pk}' = \{\hat{X}_1^{\hat{b}_1}, \hat{X}_2^{\hat{b}_2}, ..., \hat{X}_\ell^{\hat{b}_\ell}, \hat{X}_1^{\hat{d}_1\hat{b}_1}, \hat{X}_2^{\hat{d}_2\hat{b}_2}, ..., \hat{X}_\ell^{\hat{d}_\ell\hat{b}_\ell}\}$
9: $(\mathsf{pk}^{\mathcal{A}}, \sigma^{\mathcal{A}}, \mathbf{M}^{\mathcal{A}}) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}}(\cdot)}(\mathsf{pp}, \mathsf{pk}')$
10: Parse $\mathsf{pk}^{\mathcal{A}} = \{\hat{X}_1^{\mathcal{A}}, \ldots, \hat{X}_{2\ell}^{\mathcal{A}}\}$
11: Parse $\mathbf{M}^{\mathcal{A}} = \{M_1^{\mathcal{A}}, \ldots, M_{2\ell}^{\mathcal{A}}\}$
12: $\mathsf{pk}^* = \{(\hat{X}_1^{\mathcal{A}})^{1/\hat{b}_1}, \ldots, (\hat{X}_\ell^{\mathcal{A}})^{1/\hat{b}_\ell}\}$
13: $\mathbf{M}^* = \{M_{\ell+1}^{\mathcal{A}}, \ldots, M_{2\ell}^{\mathcal{A}}\}$
14: **return** $(\mathsf{pk}^*, \mathbf{M}^*, \sigma^{\mathcal{A}})$

$\mathcal{O}^{\mathsf{Sign}}(\mathbf{M}) \to \sigma$
1: Parse $\mathbf{M} = \{M_1, \ldots, M_{2\ell}\}$
2: $\sigma \leftarrow \mathcal{O}^{\mathsf{Sign},\mathsf{CL19}}(\{M_1, \ldots, M_\ell\})$
3: **return** σ

Fig. 10. Unforgeability proof reduction

secrets keys from the adversary's message vectors and public keys, thus proving Lemmas 33 and 35. We will then prove that the construction in [CL19] satisfies a modified version of public key class-hiding (Lemma 36). We will then use these extractor of messages and keys in a reduction to a challenger of this modified public key class-hiding game, using an adversary from our strong public key class-hiding game to show that our construction achieves strong public key class-hiding if the construction in [CL19] satisfies this modified public key class hiding, and thus, by Lemma 36, will prove that our scheme is secure in the generic group model (GGM). This modification of [CL19] works similarly to the game in Def. 6, but reveals the public key in the first source group (as well as the second source group). This allows our reduction to construction its public parameters in a certain way, such signatures from the [CL19] construction will either be well-formed on keys that our reduction chooses, or random keys, depending on the challenge bit. We then use this to have our strong public key class-hiding adversary distinguish these two cases.

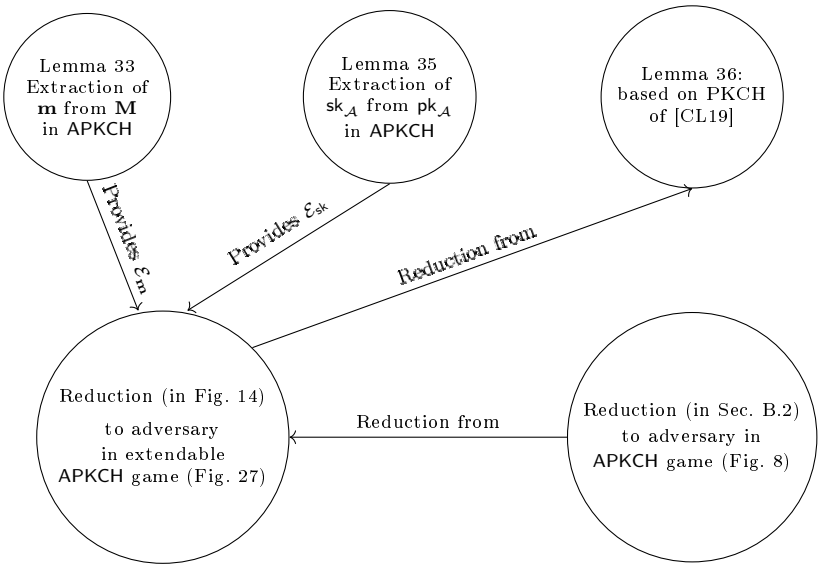

Fig. 11. Overview of the proof of Thm. 12

To make our proof simpler, we will first prove a lemma (Lemma 32) that we use in our proof.

Before we begin to write generic group model proofs, we first create a framework for proofs in the generic group model. We first define a generic group adversary in Def. 29 and informally define the GGM heuristic in Remark 30.

Definition 29 (A generic group model adversary). *A generic group model adversary is an adversary that receives a "group operation oracle" instead of knowing the exact parameters of the group which would allow the adversary to perform these operations itself. Instead, a challenger (or reduction) handles queries made to this oracle. Instead of returning group elements, the challenger returns encodings of elements. We define this oracle as* $\mathcal{O}^{\mathsf{ggm}}(\mathsf{elt}, s)$ *where* elt *is an encoding of an element and* s *is a scalar in* $\mathbb{Z}_p$ *that the adversary wishes to scale that element to (where* p *is the size of the generic group). If we are working with bilinear pairings, the adversary gets access to the oracles,* $\mathcal{O}^{\mathsf{ggm},1}(\mathsf{elt}, s)$, $\mathcal{O}^{\mathsf{ggm},2}(\mathsf{elt}, s)$, $\mathcal{O}^{\mathsf{ggm},e}(\mathsf{elt}_1, \mathsf{elt}_2)$, *and* $\mathcal{O}^{\mathsf{ggm},t}(\mathsf{elt}, s)$, *for the first source group, second source group, pairing operation, and target group, respectively. The pairing operation takes in two encodings instead of a scalar.*

Remark 30 (The GGM heuristic (informal)). A scheme proven to have certain properties against a generic group adversary has those properties in practice.

Remark 30 is false but generally holds for non-contrived schemes.

To ensure that the adversary's view is correct, the challenger will usually maintain a map from group elements to encodings which we label ϕ. At the beginning of the game, the challenger initializes this map with the parameters given to the adversary, which generally includes the generators of the bilinear pairing, P and $\hat{P}$. Thus, to begin, the challenger samples the encodings, $e_{1,P} \leftarrow \{0,1\}^\lambda$ and $e_{1,\hat{P}} \leftarrow \{0,1\}^\lambda$, and then sets $\phi(1, P) = e_{1,P}$ and $\phi(1, \hat{P}) = e_{1,\hat{P}}$. If the adversary queries $\mathcal{O}^{\mathsf{ggm},1}(P, \alpha)$, the challenger will sample and return $e_{\alpha,1}$ and set $\phi(\alpha, P) = e_{\alpha,1}$. If the adversary then samples $\mathcal{O}^{\mathsf{ggm},1}(P, 1/\alpha)$, the challenger finds that it must return $e_{1,P}$ in order to create a valid view for the adversary.

A challenger using a generic adversary can choose not to sample values in the scheme. Instead, the challenger can leave these values as *indeterminate*. For example, for DDH, instead of sampling a, b, and c, the challenger can set the maps $\phi(1, P^a) = e_{1,P^a}$, $\phi(1, P^b) = e_{1,P^b}$ and $\phi(1, P^c) = e_{1,P^c}$ without defining a, b, and c.

The usual proof strategy with the GGM is to leave all trapdoors of the scheme undefined until the end of the game and then use the Schwartz-Zippel lemma to prove that there's only a negligible chance that challenger created an invalid view.

If DDH is secure in the generic group model (which it is) our challenger can create a map, ϕ, that "works" for both cases (where $c = ab$ or $c \leftarrow\$ \mathbb{Z}_p$) i.e. the encoding is a valid view of both schemes with high probability. In other words, a generic adversary cannot distinguish the case where $c = ab$ or $c \leftarrow\$ \mathbb{Z}_p$ (the two experiments) without finding some query to the oracle that returns a different encoding in one of the games. We call this the "distinguishing query". This allows us to state Lemma Lemma 31.

Lemma 31 (Distinguishing polynomial). *For any bit-guessing security game against a generic adversary (as defined in Def. 29), either the generic adversary cannot defeat the game or there exists an extractor that can extract a "distinguishing polynomial" such that the polynomial is identically zero in one experiment and identically non-zero in the other. A bit-guessing game is defined as a game where the adversary outputs a bit and their advantage wins if they guess the challenger's random bit with greater than* $1/2 + negl(\lambda)$ *chance.*

Proof of Lemma 31. We observe that any query the adversary makes to a single oracle could instead be made to a 'polynomial' oracle. We define this polynomial oracle as such: $\mathcal{O}^{\mathsf{ggm}}(\eta, \alpha, \beta, \kappa)$ where η refers the the scalar the adversary wishes to exponentiate P by, α refers to the scalar the adversary wishes to exponentiate P^a by, β for P^b, and κ for P^c.

We call this the polynomial oracle because a reduction can represent queries as polynomials. For example, in the above query, the reduction can represent the adversary's query as $P(\eta, \alpha, \beta, \kappa) = \eta + \alpha a + \beta b + \kappa c$. The reduction can then map these polynomials to encodings.

If the adversary has an oracle in the experiment, we can update this GGM oracle. Perhaps the adversary has a signature oracle which returns a single element (σ) as the signature. We can update this oracle as the adversary makes more and more queries. After the adversary makes one signature query, the challenger updates the oracle to accept $\mathcal{O}^{\mathsf{ggm}}(\eta, \alpha, \beta, \kappa, \delta_1)$ where δ_1 is the scalar the adversary wishes to exponentiate the first signature by. After q queries, the adversary has access to the oracle, $\mathcal{O}^{\mathsf{ggm}}(\eta, \alpha, \beta, \kappa, \delta_1, \ldots, \delta_q)$.

Further, we see that this distinguishing query must be equal to a polynomial that the adversary previously queried in one experiment, and distinct from that query in the other experiment. If this is not the case, the reduction could update the map, ϕ, to make the adversary's view the same in both games.

Thus, if the adversary makes this distinguishing query to distinguish the two experiments, we find two polynomials in experiment 1, $p_{1,1}$ and $p_{2,1}$ such that $p_{2,1}(K_2) - p_{1,1}(K_1) = 0$ where K_1 and K_2 are the sets of scalars the adversary used in these queries. But, in the second experiment, $p'_{2,1}(K_2) - p'_{1,1}(K_1) \neq 0$.

Thus, we see that if the adversary can distinguish the two experiments, the challenger must be able to sift through the adversary's queries to find a polynomial in one of the experiments that is identically zero, while in the other experiment, the polynomial is not identically zero.

We use this observation in our proofs by proving that there is no such polynomial that is identically zero in one experiment while being non zero in the other. Thus, the adversary must not have been able to distinguish the two games.

As an extra note, we can see that because we only increase the options of the GGM oracle as the adversary makes new queries, any distinguishing query the adversary could've made during the game could've instead been made at the end of the game, after they've finished querying the oracles. This observation simplifies our proofs.

Lemma 32 (Extraction of m_i values from valid messages). *In the APKCH hiding game, before the challenge, because the message satisfies* $\forall i \in [\ell], e(M_i, \hat{V}_i) = e(M_{i+\ell}, \hat{V}_{i+\ell})$*, the adversary must know the discrete logs,* $\{m_i\}_{i \in [\ell]}$*, such that* $m_i = \mathsf{dlog}_{B_i}(M_i) = \mathsf{dlog}_{B_{\ell+i}}(M_{\ell+i})$ *for* $0 < i \leq \ell$.

Proof intuition for Lemma 32. We can see that if $b_i, \hat{b}_i, \hat{v}_i$ are independently random, then, verifying messages using $\hat{V}_i$ allows us to extract the m_i values in the generic group model. As a quick example to give the reader intuition as well as a review of the generic group model, we'll show this holds for messages when $\ell = 2$. Suppose an adversary supplies $M = (M_1, M_2, M_3, M_4)$ such that $e(M_1, \hat{V}_1) = e(M_3, \hat{V}_3)$ and $e(M_2, \hat{V}_2) = e(M_4, \hat{V}_4)$. We want to prove that we can extract $m_1 = \mathsf{dlog}_{B_1}(M_1) = \mathsf{dlog}_{B_{\ell+1}}(M_3)$ and $m_2 = \mathsf{dlog}_{B_2}(M_2) = \mathsf{dlog}_{B_{\ell+2}}(M_4)$. By "extract" we mean that the adversary must query these values into the GGM oracle. A generic adversary in the GGM cannot compute the group operations themselves and instead must query an oracle with scalars and encodings of group elements to perform group operations. The adversary receives encodings of the generators of the generic group, as well as encodings of any elements that the challenger creates. Thus, for pp generated for $\ell = 2$, the adversary has encodings of $P, B_1, B_2, B_{\ell+1}, B_{\ell+2}$. Ignoring the signature oracle, the adversary must construct M_1 from known encodings, i.e. the public parameters. This means they can choose a set of scalars: $K^{(1)} = (\gamma^{(1)}, \alpha_1^{(1)}, \alpha_2^{(1)}, \kappa_1^{(1)}, \kappa_2^{(1)}) \in \mathbb{Z}_p$, and query the group operation oracle for $\mathbb{G}_1$ to compute: $M_1 = P^{\gamma^{(1)}} {B_1}^{\alpha_1^{(1)}} {B_2}^{\alpha_2^{(1)}} {B_{\ell+1}}^{\kappa_1^{(1)}} {B_{\ell+2}}^{\kappa_2^{(1)}}$. Let us also label their computation of M_3 as $M_3 = P^{\gamma^{(3)}} {B_1}^{\alpha_1^{(3)}} {B_2}^{\alpha_2^{(3)}} {B_{\ell+1}}^{\kappa_1^{(3)}} {B_{\ell+2}}^{\kappa_2^{(3)}}$. Because $e(M_1, \hat{V}_1) = e(M_3, \hat{V}_3)$ (and $\mathsf{dlog}_P(\hat{V}_1) = (\hat{v}_1\hat{b}_1)$ and $\mathsf{dlog}_P(\hat{V}_3) = (\hat{v}_1)$) we know that:

$$\begin{aligned}&(\gamma^{(1)} + b_1\alpha_1^{(1)} + b_2\alpha_2^{(1)} + (b_1\hat{b}_1)\kappa_1^{(1)} + (b_2\hat{b}_2)\kappa_2^{(1)}) * \hat{v}_1\hat{b}_1 \\ &\quad = (\gamma^{(3)} + b_1\alpha_1^{(3)} + b_2\alpha_2^{(3)} + (b_1\hat{b}_1)\kappa_1^{(3)} + (b_2\hat{b}_2)\kappa_2^{(3)}) * \hat{v}_1 \end{aligned} \tag{2}$$

We can simplify this by dividing both sides by $1/(b_1\hat{b}_1)$:

$$\begin{aligned}&(\gamma^{(1)} + b_1\alpha_1^{(1)} + b_2\alpha_2^{(1)} + (b_1\hat{b}_1)\kappa_1^{(1)} + (b_2\hat{b}_2)\kappa_2^{(1)}) * \hat{v}_1/b_1 \\ &\quad = (\gamma^{(3)} + b_1\alpha_1^{(3)} + b_2\alpha_2^{(3)} + (b_1\hat{b}_1)\kappa_1^{(3)} + (b_2\hat{b}_2)\kappa_2^{(3)}) * \hat{v}_1/(b_1\hat{b}_1) \end{aligned} \tag{3}$$

Distibuting Eq. 3 we get Eq 4:

$$\begin{aligned}&(\gamma^{(1)}\hat{v}_1/b_1 + \alpha_1^{(1)}\hat{v}_1 + b_2\alpha_2^{(1)}\hat{v}_1/b_1 + \hat{b}_1\kappa_1^{(1)}\hat{v}_1 + b_2\hat{b}_2\kappa_2^{(1)}\hat{v}_1\hat{b}_1) \\ &\quad = (\gamma^{(3)}\hat{v}_1/(b_1\hat{b}_1) + \alpha_1^{(3)}\hat{v}_1/\hat{b}_1 + b_2\alpha_2^{(3)}\hat{v}_1/(b_1\hat{b}_1) + \kappa_1^{(3)}\hat{v}_1 + b_2\hat{b}_2\kappa_2^{(3)}\hat{v}_1/(b_1\hat{b}_1)) \end{aligned} \tag{4}$$

This implies that when you subtract the right side of Eq. 4 from the left, it equals 0. The general proof strategy in the GGM is to leave values that the adversary does not know undefined and instead have the oracle remember algebraic relations between encodings to return encodings such that the adversary will not notice that the challenger has not initialized the undefined variables. We call these undefined variables "indeterminate". The oracle then samples the indeterminate after the game is complete and proves that there is a negligible chance that the adversary's scalar operations resulted in winning the security game (e.g. creating a forgery). We can see that every term on the right side of Eq. 4 includes indeterminate $1/\hat{b}_1$ except for $\hat{v}_1\kappa_1^{(3)}$. On the left side, every term has $1/b_1$ or $\hat{b}_1$ except for $\hat{v}_1\alpha_1^{(1)}$. Thus, because $\hat{b}_1$ and b_1 are indeterminate, the only way for the adversary to satisfy this relation in the GGM after the GGM oracle instantiates variables b_1 and $\hat{b}_1$ randomly is to set $\kappa_1^{(3)} = \alpha_1^{(1)}$ and leave all other chosen values 0. This gives us $M_1 = {B_1}^{\alpha_1^{(1)}}$ and $M_3 = {B_{\ell+1}}^{\kappa_1^{(3)}}$, allowing a reduction to extract an m_1 value that satisfies $M_1 = {B_1}^{m_1}$ and $M_3 = {B_{\ell+1}}^{m_1}$. While the $\hat{v}_1$ and $\hat{v}_2$ values seem unnecessary here, they ensure that an adversary that knows m_1, m_2 cannot use the bases $(\hat{V}_i)$ to recognize a message after it has been randomized. We now introduce the formal proof of Lemma 32.

Proof of Lemma 32. To prove this, we'll first show that using the public parameters alone, a generic adversary cannot produce a message where they do not know the correct discrete log.

The adversary sees $\mathsf{pp} = \{B_i, \hat{B}_i, B_{\ell+i}, \hat{B}_{\ell+i}, V_i, V_{i+\ell}, \hat{V}_i, \hat{V}_{i+\ell}\}_{i\in[\ell]}$, where $\forall i \in [\ell], B_i \leftarrow P^{b_i}, \hat{B}_i \leftarrow \hat{P}^{\hat{b}_i}, B_{\ell+i} \leftarrow P^{b_i\hat{b}_i}, \hat{B}_{\ell+i} \leftarrow \hat{P}^{\hat{b}_i\hat{d}_i}$, $\hat{V}_i = \hat{P}^{\hat{v}_i/b_i}$, $\hat{V}_{i+\ell} = \hat{P}^{\hat{v}_i/b_i\hat{b}_i}$, $V_i = P^{v_i/\hat{b}_i}$, $V_{i+\ell} = P^{v_i/b_i\hat{b}_i}$.

The adversary sees $\{B_i, B_{\ell+i}, V_i, V_{i+\ell}\}_{i\in[\ell]}$ in $\mathbb{G}_1$ and can exponentiate these values to create a message. Thus, we can represent any element in $\mathbb{G}_1$ that the adversary can create with the following polynomial: $Q(\{\alpha_i, \kappa_i, \nu_i, \nu_{i+\ell}\}_{i\in[\ell]}) = \sum_{i\in[\ell]} \alpha_i b_i + \sum_{i\in[\ell]} \kappa_i b_i \hat{b}_i + \sum_{i\in[\ell]} \nu_i v_i/\hat{b}_i + \sum_{i\in[\ell]} \nu_{i+\ell} v_i/(b_i\hat{b}_i)$, where $\{\alpha_i, \kappa_i, \nu_i,\ \nu_{i+\ell}\}_{i\in[\ell]}$ can be chosen by the adversary. We'll look at one verification equation for M_j and $M_{j+\ell}$ and label $Q(K)$ where $K = \{\alpha_i, \kappa_i, \nu_i, \nu_{i+\ell}\}_{i\in[\ell]}$ as the computation of M_j and $Q'(K')$ where $K' = \{\alpha'_i, \kappa'_i, \nu'_i, \nu'_{i+\ell}\}_{i\in[\ell]}$ as the computation for $M_{j+\ell}$. We see that because $e(M_j, \hat{V}_j) = e(M_{j+\ell}, \hat{V}_{j+\ell})$, we must have that $\hat{v}_j Q(K)/b_j = \hat{v}_j Q'(K')/(b_j\hat{b}_j)$. We can multiply this equation on either side by $b_i\hat{b}_i$ to we see that $Q(K)\hat{b}_j = Q'(K')$. Thus, it must be that:

$$\sum_{i\in[\ell]} \alpha_i b_i \hat{b}_j + \sum_{i\in[\ell]} \kappa_i b_i \hat{b}_i \hat{b}_j + \sum_{i\in[\ell]} \nu_i v_i \hat{b}_j/\hat{b}_i + \sum_{i\in[\ell]} \nu_{i+\ell} v_i \hat{b}_j/(b_i\hat{b}_i) = \sum_{i\in[\ell]} \alpha'_i b_i + \sum_{i\in[\ell]} \kappa'_i b_i \hat{b}_i + \sum_{i\in[\ell]} \nu'_i v_i/\hat{b}_i + \sum_{i\in[\ell]} \nu'_{i+\ell} v_i/(b_i\hat{b}_i)$$

We can quickly see that these polynomials will certainly not be equal if we have any $\alpha_i, \kappa_i, \nu_i, \nu_{i+\ell} \neq 0$ where $i \neq j$ since in this case, the left equation would have some $b_i\hat{b}_j$ or $\hat{b}_j/\hat{b}_i$ term which the right equation does not. From this, we can deduce that for all $i \neq j$, $\alpha'_i, \kappa'_i, \nu'_i, \nu'_{i+\ell} = 0$, since we established that for any term where $i \neq j$, the left side must be zero and thus including any non-zero indeterminate on the right side of the equation would unbalance the equation.

Thus, we have that only the bases, $\{B_j, B_{\ell+j}, V_j, V_{\ell+j}\}$ are used in the computation of $M_j, M_{j+\ell}$ for any j (critically, $\{B_i, B_{\ell+i}, V_i, V_{\ell+i}\}_{i\in[\ell]\setminus\{j\}}$ are excluded from the computation). Thus, we have the following equation for the verification of M_j and $M_{j+\ell}$:

$$\alpha_j b_j \hat{b}_j + \kappa_j b_j \hat{b}_j^2 + \nu_j v_j + \nu_{j+\ell} v_j/(b_j) = \alpha'_j b_j + \kappa'_j b_j \hat{b}_j + \nu'_j v_j/\hat{b}_j + \nu'_{j+\ell} v_j/(b_j\hat{b}_j)$$

From this, we can see that this is only satisfied when $\alpha_j = \kappa'_j$ and all other values chosen by the adversary are 0. Thus, we can extract $m_i = \alpha_j = \kappa'_j = \mathsf{dlog}_{B_i}(M_i) = \mathsf{dlog}_{B_{\ell+i}}(M_{i+\ell})$.

Next, we need to show that even after seeing signatures, the adversary cannot include secrets from the signatures in their messages. Since the adversary cannot create a message without knowing the discrete log using the parameters, and the public key is in $\mathbb{G}_2$, we know the message in the first query to the signing oracle is independent of any of the challenger's secret. Now, we'll assume this for query n and show that for $n+1$ this holds (proof by induction).

For query j, the adversary chooses a message, $\{\mu_{i,j}\}_{i\in[\ell]}$, and the challenger signs it, yielding $Z_j = \prod_{i,j} P^{\mu_{i,j} b_i \hat{b}_i x_i y_j}, Y_j = P^{1/y_j}$ in $\mathbb{G}_1$. After query n, the adversary has seen $\{Z_j, Y_j\}_{j\in[n]}$ in $\mathbb{G}_1$ along with the public parameters. Let K be the set of the adversary's choices. $K = \{\mu_{i,j}, \gamma_j, \alpha_i, \kappa_i, \nu_i, \nu_{i+\ell}\}_{j\in[n],i\in[\ell]}$ where $\alpha_i, \kappa_i, \nu_i, \nu_{i+\ell}$ are defined similar to their previous definition in this proof, $\mu_{i,j}$ is the choice of message, and γ_j is the scalar for Y_j. We can update the polynomial representing any of the adversary's evaluations in $\mathbb{G}_1$ from the last proof (when the adversary only had pp). We'll split this polynomial (R) into the sum of two polynomial: Q (from the last proof) and S using signatures (new to this proof). $R(K) = Q(K_Q) + S(K_S)$ where $K_Q = \{\alpha_i, \kappa_i, \nu_i, \nu_{i+\ell}\}_{i\in[\ell]}$ and $K_S = \{\mu_{i,j}, \gamma_j\}_{i\in[\ell],j\in[n]}$. S is defined as $S(K_S) = \sum_{j\in[n]}(\sum_{i\in[\ell]} \mu_{i,j} b_i \hat{b}_i x_i y_j) + \sum_{j\in[n]} \gamma_j/y_j$. We do not include a direct scalar for exponentiating Z_j values because any scaling of Z_j by the adversary can be countered by multiplying with $\mu_{i,j}$, as the relation is proportional.

Again, focusing on one verification equation for index k:

If $e(M_k, \hat{V}_k) = e(M_{k+\ell}, \hat{V}_{k+\ell})$ we see that $\hat{b}_k R(K) = R'(K')$ (where $R(K)$ is the adversary's computation of M_k and $R'(K')$ is the computation of $M_{k+\ell}$). Because S is dependent on y_j which does not appear in Q, no term in Q can be used to cancel these terms out. Thus, it must be that: $\hat{b}_k S(K_S) = S'(K'_S)$. This means that:

$$\sum_{j\in[n]}(\sum_{i\in[\ell]} \mu_{i,j} \hat{b}_k b_i \hat{b}_i x_i y_j) + \sum_{j\in[n]} \gamma_j \hat{b}_k/y_j = \sum_{j\in[n]}(\sum_{i\in[\ell]} \mu'_{i,j} b_i \hat{b}_i x_i y_j) + \sum_{j\in[n]} \gamma'_j/y_j$$

Where $K'_S = \{\mu'_{i,j}, \gamma'_j\}_{i\in[\ell],j\in[n]}$ are the adversary's choices from K' used in the computation of $Q'(K'_S)$ used to compute $M_{k+\ell}$. We can see that all terms on the left side of the equation include the indeterminate $\hat{b}_k$ whereas on the right side, only one includes $\hat{b}_k$. Thus, it must be that:

$$\sum_{j\in[n]}(\sum_{i\in[\ell]} \mu_{i,j} \hat{b}_k b_i \hat{b}_i x_i y_j) + \sum_{j\in[n]} \gamma_j \hat{b}_k/y_j = (\sum_{i\in[\ell]} \mu'_{i,j} b_i \hat{b}_k x_i y_k) + \gamma'_k/y_k$$

We can see that on the left side, each term includes some y_j when $j \neq k$ and thus it must be that:

$(\sum_{i\in[\ell]} \mu_{i,j}\hat{b}_k b_i \hat{b}_i x_i y_k) + \gamma_k \hat{b}_k / y_k = (\sum_{i\in[\ell]} \mu'_{i,j} b_i \hat{b}_k x_i y_k) + \gamma'_k / y_k$

We can see that terms on the left side include either $\hat{b}_k\hat{b}_i$ or $\hat{b}_k/y_k$ while indeterminate of this form are absent on the right side. Thus, it must be that $\forall i \in [\ell], j \in [n], \mu_{i,j} = 0, \gamma_j = 0$.

Thus, after n queries to the oracle, the adversary must still know the discrete log of the message, and thus by induction, must always know the discrete logs of messages during the game. □

Recall (see Fig. 2) the security game. After the adversary interacts with the signing oracle for pk for a while, it is ready to receive its challenge: a public key pk^b that is either in the same equivalence class as pk or in the same class as $\mathsf{pk}_{\mathcal{A}}$ chosen by the adversary; as well as the signature σ^b under pk^b on the message M of the adversary's choice. The adversary may continue making further queries to the signing oracle for the original public key pk. After receiving $(\mathsf{pk}^b, \sigma^b)$, the adversary may be able to use the fact that, in case $b = 1$, the signature σ^b will include the adversary's own secrets. For example, the adversary can incorporate these secrets into the messages they choose to query to the signing oracle. Thus, we need to prove a second lemma to ensure that the adversary does not include elements of previous signatures in their queries after receiving the challenge.

Lemma 33 (Extraction of m_i values from valid messages after the challenge). *In the APKCH hiding game, because the message satisfies $\forall i \in [\ell], e(M_i, \hat{V}_i) = e(M_{i+\ell}, \hat{V}_{i+\ell})$, the adversary must know the discrete logs, $\{m_i\}_{i\in[\ell]}$, such that $m_i = \mathsf{dlog}_{B_i}(M_i) = \mathsf{dlog}_{B_{\ell+i}}(M_{\ell+i})$ for $0 < i \leq \ell$.*

Proof of Lemma 33. We can apply our logic from the proof of Lemma 32 to ensure that when $b = 0$ the adversary cannot make their messages depend on secrets from the signature since the returned challenge will be distributed identically to other queries. This is because the challenge signature comes from the same oracle that the adversary has already been receiving signatures on, it is just randomized. Thus, fixing $b = 0$, if an adversary exists that can break key extraction with the challenge, we can reduce to a reduction that does not receive this challenge, but instead simply queries one more signature, randomizes it, and hands it to this adversary as the challenge.

When $b = 1$, the adversary gains a values in $\mathbb{G}_1$ which are distinct from previous signatures. Specifically, they learn: $Z_c = P^{\sum_{i\in[\ell]} \mu_{i,c}\rho_c\chi_i y_c}$ and $Y_c = P^{1/y_c}$ where $\{\chi_i\}_{i\in[\ell]}$ is the adversary's secret key, $\{\mu_{i,c}\}_{i\in[\ell]}$ is the adversary's submitted message, ρ_c is the randomization that the challenger computes, and y_c is the randomizer used in the signature. As in the proof of Lemma 32, we split the polynomials representing M_k and $M_{k+\ell}$ into sums of polynomials Q and S. But, we include Z_c in the computation of S such that for $K_S = \{\mu_{i,c}, \gamma_c, \mu_{i,j}, \gamma_j\}_{i\in[\ell],j\in[n]}$, $S(K_S) = \sum_{j\in[n]}(\sum \mu_{i,j} b_i \hat{b}_i x_i y_j) + \sum_{j\in[n]} \gamma_j/y_j + (\sum \mu_{i,c} b_i \hat{b}_i \rho_c y_c) + \gamma_c/y_c$. Again, because the adversary could simply compute $\mu'_{i,c} = \chi_i \mu_{i,c}$, we do not include χ_i in this polynomial WLOG.

Again, we see that all terms in $S(K_S)$ include y_c or y_j and thus, no term in $Q(K_Q)$ can cancel these indeterminate out. Thus it must be that $S(K_S)\hat{b}_k = S(K'_s)$

$\sum_{j\in[n]}(\sum \mu_{i,j}\hat{b}_k b_i \hat{b}_i x_i y_j) + \sum_{j\in[n]} \gamma_j \hat{b}_k / y_j + (\sum \mu_{i,c}\hat{b}_k b_i \hat{b}_i \rho_c y_c) + \gamma_c \hat{b}_k / y_c$
$= \sum_{j\in[n]}(\sum \mu'_{i,j} b_i \hat{b}_i x_i y_j) + \sum_{j\in[n]} \gamma'_j / y_j + (\sum \mu'_{i,c} b_i \hat{b}_i \rho_c y_c) + \gamma'_c / y_c$

Using similar logic from the proof of Lemma 32 we can see that $\forall i \in [\ell], j \in [n], \mu_{i,j} = 0, \mu_{i,c} = 0, \gamma_j = 0$ and $\gamma_c = 0$. The logic from the proof of Lemma 32 holds because it only relies on the presence of $\hat{b}_k$ and not on x_i which is the only difference that the challenge signature has. □

Before we continue, we need to prove that we can extract the secret key from the adversary's public key in the experiment in Fig. 2. We define this property in Def. 34.

Definition 34 (Key extractability). *For all parameters λ, ℓ and any PPT algorithm, $\mathcal{A}$, there exists an efficient extraction algorithm, $\mathcal{E}_{\mathsf{sk}}$ and negligible function, negl such that:*

$$\Pr\left[\begin{array}{l|l} \mathsf{VerifyKey}(\mathsf{pp}, \mathsf{pk}^{\mathcal{A}}) = 1 \wedge & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell); (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(\mathsf{pp}); \\ (\mathsf{sk}^{\mathcal{A}}, \mathsf{pk}^{\mathcal{A}}) \notin \mathsf{KGen}(\mathsf{pp}) & \mathsf{pk}^{\mathcal{A}} \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{pp}); \mathsf{sk}^{\mathcal{A}} \leftarrow \mathcal{E}_{\mathsf{sk}}(\mathsf{pk}^{\mathcal{A}}) \end{array}\right] \leq negl(\lambda)$$

Lemma 35 (Extraction of the secret key (x_i) values from valid public keys). *The mercurial signature construction in Fig. 3 has key extraction as defined in Def. 34.*

Proof of Lemma 35. Intuitively, in the APKCH hiding game, because the public key satisfies $\forall i \in [\ell], e(V_i, \hat{X}_i) = e(V_{i+\ell}, \hat{X}_{i+\ell})$, the adversary must know the discrete logs, $\{x_i\}_{i\in[\ell]}$, such that $x_i = \mathsf{dlog}_{\hat{B}_i}(\hat{X}_i) = \mathsf{dlog}_{\hat{B}_{\ell+i}}(\hat{X}_{\ell+i})$ for $0 < i \leq \ell$.

More formally, the adversary sees $\mathsf{pp} = \{B_i, \hat{B}_i, B_{\ell+i}, \hat{B}_{\ell+i}, V_i, V_{i+\ell}, \hat{V}_i, \hat{V}_{i+\ell}\}_{i\in[\ell]}$, where $\forall i \in [\ell], B_i \leftarrow P^{b_i}, \hat{B}_i \leftarrow \hat{P}^{\hat{b}_i}, B_{\ell+i} \leftarrow P^{b_i\hat{b}_i}, \hat{B}_{\ell+i} \leftarrow \hat{P}^{\hat{b}_i\hat{d}_i}, \hat{V}_i = \hat{P}^{\hat{v}_i\hat{b}_i}, \hat{V}_{i+\ell} = \hat{P}^{\hat{v}_i}, V_i = P^{v_i\hat{d}_i}, V_{i+\ell} = P^{v_i}$.

After q queries to the signature oracle, the GGM adversary will create the following polynomials in $\mathbb{G}_2$:

$$P(K) = \sum \beta_i\hat{b}_i + \sum \beta_{\ell+i}\hat{b}_i\hat{d}_i + \sum \gamma_j y_j + \nu_i\hat{v}_i\hat{b}_i + \sum \nu_i\hat{v}_i$$

Where K is a set of the adversary's choice of β_i, γ_j, and ν_i.

Because the adversary's public key verifies, i.e. $e(V_i, \hat{X}_i) = e(V_{\ell+i}, \hat{X}_{\ell+i})$, we know that:

$$\hat{d}_i P(K_i) = P(K_{\ell+i})$$

Thus, $P(K_{\ell+i})$ must contain $\hat{d}_i$ or $P(K)$ must be zero, which can be easily detected and rejected. We can see that the only term in $P(K)$ that includes $\hat{d}_i$ is $\beta_{\ell+i}\hat{b}_i\hat{d}_i$. Thus, the adversary must use this term in the computation of $P(K_{\ell+i})$.

We can see that this includes the indeterminate $\hat{b}_i$. Thus, the adversary must include the terms $\beta_i\hat{b}_i$ or $\nu_i\hat{v}_i\hat{b}_i$ in the computation of $P(K_i)$. If the adversary were to include $\nu_i\hat{v}_i\hat{b}_i$ in the computation of $P(K_i)$, we can see that no other term includes $\hat{d}_i\hat{v}_i\hat{b}_i$. Thus, there would be no way for the adversary to balance this equation. Therefore, the adversary must only use the term $\beta_i\hat{b}_i$ for $P(K_i)$ and $\beta_{\ell+i}\hat{b}_i\hat{d}_i$ for $P(K_{\ell+i})$, meaning we can extract these values as the correct secret key.

After the challenge, the adversary only learns a single additional element in $\mathbb{G}_2$, $\hat{Y} = \hat{P}^{1/y}$. The discrete log of this element is independent of any other elements that the adversary saw in $\mathbb{G}_2$ so it does not help the adversary incorporate it into their computation of any verifying public key.

Proof of Thm. 12. In proving Thm. 12, we'll see that Lemma 36 will become useful. This game in this lemma is similar to PKCH of the construction in [CL19] but does not give the adversary oracles and instead generates a signature on a single message (where the adversary knows the message itself). This game is similar to the public key class-hiding game in [CL19], but the adversary receives some structured bases as $\mathbf{B^l}, \mathbf{B^u}$, and $\hat{\mathbf{B}}^{\mathbf{l}}$ are related in a DDH type of way. They also effectively can only query the oracle once, which is modeled by having the adversary output the message that they want signed and then calling them with the signature later. Later, our reduction will play the role of the adversary in this experiment.

Lemma 36. *For any PPT generic adversary, $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$, we have that* $\left|\Pr\left[\boldsymbol{Exp}^{\mathsf{PKCH-Prf},0}_{\mathcal{A}} = 1\right] - \Pr\left[\boldsymbol{Exp}^{\mathsf{PKCH-Prf},1}_{\mathcal{A}} = 1\right]\right|$ *is negligible. Where the experiment* $\boldsymbol{Exp}^{\mathsf{PKCH-Prf},\mathsf{b}}_{\mathcal{A}}$ *is described in Fig. 12.*

$\{b_i, \hat{b}_i, e_i\}_{i\in[\ell]}, \rho, y \leftarrow\$ \mathbb{Z}_p$
$\mathbf{B^l} = \{B_i\}_{i\in[\ell]}$ where $B_i = P^{b_i}$
$\mathbf{B^u} = \{B_{\ell+i}\}_{i\in[\ell]}$ where $B_{\ell+i} = P^{b_i\hat{b}_i}$
$\hat{\mathbf{B}}^{\mathbf{l}} = \{B_i\}_{i\in[\ell]}$ where $\hat{B}_i = \hat{P}^{\hat{b}_i}$
$\hat{\mathbf{E}}^{\mathbf{0}} = \{E_i^0\}_{i\in[\ell]}$ where $\hat{E}_i^0 = \hat{P}^{\rho\hat{b}_i}$
$\hat{\mathbf{E}}^{\mathbf{1}} = \{E_i^1\}_{i\in[\ell]}$ where $\hat{E}_i^1 = \hat{P}^{e_i}$
$\mathbf{m} \leftarrow \mathcal{A}_1(\mathbf{B^l}, \mathbf{B^u}, \hat{\mathbf{B}}^{\mathbf{l}}, \hat{\mathbf{E}}^{\mathbf{b}})$
$Z^0 = (\prod P^{m_i\rho\hat{b}_i})^y, Y^0 = P^{1/y}, \hat{Y}^0 = \hat{P}^{1/y}$
$Z^1 = (\prod P^{m_i e_i})^y, Y^1 = P^{1/y}, \hat{Y}^1 = \hat{P}^{1/y}$
$b' \leftarrow \mathcal{A}_2(Z^b, Y^b, \hat{Y}^b)$
return b'

Fig. 12. Experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{PKCH\text{-}Prf},b}$ used in Proof of Thm. 12.

Proof of Lemma 36. We will prove this in the generic group model. In the generic group model, the goal of any adversary is to find a computation that is equivalent to $e(P, \hat{P})$ when $b = b'$ but is not equal to $e(P, \hat{P})$ when $b = 1 - b'$.

Lemma 37. *If the adversary does not use $B_{\ell+i}$ values in their distinguishing polynomial in Fig. 12, we can reduce to PKCH of the mercurial signature construction in [CL19] (distinguishing polynomials are defined in Lemma 31).*

Proof of Lemma 37. Intuitively, this Lemma holds because $B_{\ell+i}$ is the only element that differs from the PKCH game and everything else, a reduction can generate. This can be seen by looking at computations that the adversary can create without these elements (disregarding Z for now):
$e(P,\hat{P})^{\eta} \prod e(P,\hat{B}_i)^{\beta_i^l} \prod e(P,\hat{E}_i^b)^{\epsilon_i} \prod e(B_i,\hat{P})^{\alpha_i^l} \prod e(B_i,\hat{B}_j)^{\gamma_{\beta,j}^{\alpha,i}} \prod e(B_i,\hat{E}_j^b)^{\gamma_{\epsilon,j}^{\alpha,i}}$
This polynomial appears as:
$Q(K) = \eta + \sum \hat{b}_i * \beta_i^l + \sum e_i * \epsilon_i + \sum b_i * \alpha_i^l + \sum b_i * \hat{b}_j * \gamma_{\beta,j}^{\alpha,i} + \sum b_i * e_j * \gamma_{\epsilon,j}^{\alpha,i}$
We can create a reduction playing the PKCH game with the CL19 construction which generates these trapdoors, b_i itself. Because we are assuming that $B_{\ell+i}$ is never included in any distinguishing polynomial, our reduction does not need to know $\hat{b}_i$ to create encoding when the adversary exponentiates it. Instead, the reduction simply leaves the discrete log of each $B_{\ell+i}$ as an indeterminate and maps any encoding of this element to be the same as $e(B_i, \hat{B}_j)$. After this adversary finds a distinguishing polynomial, our reduction removes their own trapdoors from the polynomial, leaving:
$Q'(K) = \eta + \sum \hat{b}_i * \beta_i^l + \sum e_i * \epsilon_i + \sum \alpha_i^l + \sum \hat{b}_j * \gamma_{\beta,j}^{\alpha,i} + \sum e_j * \gamma_{\epsilon,j}^{\alpha,i}$
We can see that if $Q(K)$ is a distinguishing polynomial in the game in Fig. 12, then $Q'(K)$ is a distinguishing polynomial in the CL19 game when we replace $\hat{B}_{\ell+i}$ with elements from the pk_1 from the CL19 challenger and replace $\hat{E}_i^b$ with elements from the pk_2^b challenger. Thus, the adversary must include one of these elements ($B_{\ell+i}$ or Z) in their distinguishing polynomial, otherwise, we break the PKCH of the construction in CL19. We have a similar argument to Lemma 37 for proving that the distinguishing polynomial cannot be independent of $B_{\ell+i}$ as our reduction can generate Z^b using the CL19 challenger. □

Thus, we find that the distinguishing polynomial must include $B_{\ell+i}$ (Lemma 37).

The only elements that have b_i in them are B_i and $B_{\ell+i}$. Thus, for the adversary to win, they must be able to cancel out the $B_{\ell+i}$ that they include in their computation with one of these elements.

The adversary must also include the Z or $\hat{E}_i^b$ elements as these are the only elements that differ in the two games.

We can see that (without using Z^b) any computation with Y or $\hat{Y}$ will only add the indeterminate $1/y$ to the computation, any these could simply be replaced with P with the same result. So, we can ignore these for now.

Because the Z^b values are independent of the b_i values, they cannot be used to cancel out the given $B_{\ell+i}$. These might still be useful if the adversary is able to first cancel out the b_i present in their

computation of $B_{\ell+i}$. Thus, if we can first prove that the adversary cannot cancel out the b_i in their computation without the Z^b values, there's nothing the adversary can do to use these Z^b values to obtain an equation that results in $e(P, \hat{P})$.

In a similar argument, no computation without B_i or $B_{\ell+i}$ can be used to cancel out the b_i values.

Thus, the part of the adversary's computation that cancels out b_i appears as the following equation where the adversary can choose to include or remove specific pairings:

$\prod e(B_i, \hat{P})^{\gamma^{\alpha,i}} \prod e(B_i, \hat{B}_j)^{\gamma^{\alpha,i}_{\beta,j}} \prod e(B_i, \hat{E}^b_j)^{\gamma^{\alpha,i}_{\eta,j}} \prod e(B_{\ell+i}, \hat{P})^{\gamma^{\kappa,i}} \prod e(B_{\ell+i}, \hat{B}_j)^{\gamma^{\kappa,i}_{\beta,j}}$
$\prod e(B_{\ell+i}, \hat{E}^b_j)^{\gamma^{\kappa,i}_{\eta,j}}$

In order to cancel out an b_i, there must be some division of elements with this value. For any B_i or $B_{\ell+i}$ used in the numerator of the computation, there must be another in the denominator.

Thus, $\forall i$, $\gamma^{\alpha,i} + \gamma^{\alpha,i}_{\beta,j} + \gamma^{\alpha,i}_{\eta,j} + \gamma^{\kappa,i} + \gamma^{\kappa,i}_{\beta,j} + \gamma^{\kappa,i}_{\eta,j} = 0$

Let's look at the discrete log of this equation:

If $b = 0$:
$\sum \gamma^{\alpha,i} b_i + \sum \gamma^{\alpha,i}_{\beta,j} b_i \hat{b}_j + \sum \gamma^{\alpha,i}_{\eta,j} b_i \rho \hat{b}_j + \sum \gamma^{\kappa,i} b_i \hat{b}_i + \sum \gamma^{\kappa,i}_{\beta,j} b_i \hat{b}_i \hat{b}_j + \sum \gamma^{\kappa,i}_{\eta,j} b_i \hat{b}_i \rho \hat{b}_j$

If $b = 1$:
$\sum \gamma^{\alpha,i} b_i + \sum \gamma^{\alpha,i}_{\beta,j} b_i \hat{b}_j + \sum \gamma^{\alpha,i}_{\eta,j} b_i e_j + \sum \gamma^{\kappa,i} b_i \hat{b}_i + \sum \gamma^{\kappa,i}_{\beta,j} b_i \hat{b}_i \hat{b}_j + \sum \gamma^{\kappa,i}_{\eta,j} b_i \hat{b}_i e_j$

We can see that if the adversary sets $\gamma^{\alpha,i} > 0$, they have a $e(P, \hat{P})^{b_i}$ term. Because each other term has $\hat{b}_i$, this cannot be removed from the polynomial. Thus, it must be that $\gamma^{\alpha,i} = 0$.

Thus, we have the two computations:

If $b = 0$:
$\sum \gamma^{\alpha,i}_{\beta,j} b_i \hat{b}_j + \sum \gamma^{\alpha,i}_{\eta,j} b_i \rho \hat{b}_j + \sum \gamma^{\kappa,i} b_i \hat{b}_i + \sum \gamma^{\kappa,i}_{\beta,j} b_i \hat{b}_i \hat{b}_j + \sum \gamma^{\kappa,i}_{\eta,j} b_i \hat{b}_i \rho \hat{b}_j$

If $b = 1$:
$\sum \gamma^{\alpha,i}_{\beta,j} b_i \hat{b}_j + \sum \gamma^{\alpha,i}_{\eta,j} b_i e_j + \sum \gamma^{\kappa,i} b_i \hat{b}_i + \sum \gamma^{\kappa,i}_{\beta,j} b_i \hat{b}_i \hat{b}_j + \sum \gamma^{\kappa,i}_{\eta,j} b_i \hat{b}_i e_j$

It must be that some $\gamma^{\alpha,i}_{\eta,j}$, $\gamma^{\kappa,i}_{\eta,j}$ must be non-zero otherwise the equation would always look the same in both games.

These contain either ρ or e_j which are not included in the other terms. Thus, they can only be canceled out by each other and thus, the adversary must be able to cancel them out with the following computations:

If $b = 0$: $\sum \gamma^{\alpha,i}_{\eta,j} b_i \rho \hat{b}_j + \sum \gamma^{\kappa,i}_{\eta,j} b_i \hat{b}_i \rho \hat{b}_j$

If $b = 1$: $\sum \gamma^{\alpha,i}_{\eta,j} b_i e_j + \sum \gamma^{\kappa,i}_{\eta,j} b_i \hat{b}_i e_j$

We can see that the second term in $b = 0$ contains $\hat{b}_i \hat{b}_j$, which the first term does not. Thus, this polynomial must be zeroed out on its own. Similarly for when $b = 1$, the second term contains $\hat{b}_i$ while the first term does not. Thus, these terms must zero out independently of one another. Thus, we can analyze them independently.

Case 1:

If $b = 0$: $\sum \gamma^{\alpha,i}_{\eta,j} b_i \rho \hat{b}_j$

If $b = 1$: $\sum \gamma^{\alpha,i}_{\eta,j} b_i e_j$

Case 2:

If $b = 0$: $\sum \gamma^{\kappa,i}_{\eta,j} b_i \hat{b}_i \rho \hat{b}_j$

If $b = 1$: $\sum \gamma^{\kappa,i}_{\eta,j} b_i \hat{b}_i e_j$

We'll deal with case 1 first. We see that any b_i cannot be cancelled out for any term that includes a different b_j. Thus, the computation for a fixed i must be equal to $e(P, \hat{P})$. We can see that

Thus, for a given i,

If $b = 0$: $\sum_j \gamma_{\eta,j}^{\alpha,i} b_i \rho \hat{b}_j = e(P, \hat{P})$

If $b = 1$: $\sum_j \gamma_{\eta,j}^{\alpha,i} b_i e_j = e(P, \hat{P})$

We can see that each of these terms have a distinct $\hat{b}_j$ or e_j and thus they can only be zero when $\gamma_{\eta,j}^{\alpha,i} = 0$ for all j.

Moving on to case 2, we can use the same logic from case 1 to fix an i value.

For a given i,

If $b = 0$: $\sum_j \gamma_{\eta,j}^{\kappa,i} b_i \hat{b}_i \rho \hat{b}_j$

If $b = 1$: $\sum_j \gamma_{\eta,j}^{\kappa,i} b_i \hat{b}_i e_j$

We can see in the second game that each term has a distinct e_j in them and thus will never cancel out. In the first game, we see that either there is a distinct $\hat{b}_j$ in the term, or a $(\hat{b}_i)^2$ when $i = j$. In both cases, the polynomial does not zero out.

Thus, the adversary cannot zero out one polynomial in one game without zeroing out the other game and due to Lemma 31, the adversary has no chance at distinguishing in the generic group model. □.

Proof of Thm. 12 using Lemma 36. We can now reduce to Lem. 36 to ensure that our scheme in Fig. 3 satisfies APKCH.

To do this, we'll create three hybrids. The first hybrid is where the challenge signature (σ^b) returned in the APKCH game (Fig. 2) is the same one used in the signature oracle in this game (pk_0). The second hybrid is where the challenge signature returned is from a new random key, independent of pk_0 and $\mathsf{pk}_{\mathcal{A}}$. The third hybrid is where the returned challenge secret key is from a randomization of the adversary's public key $\mathsf{pk}_{\mathcal{A}}$.

We'll first prove that the first two hybrids are indistinguishable. The reduction will generate the sk used in (Fig. 2). To first prove that this reduction secret key is indistinguishable from random in the challenge key/signature, the reduction will specially craft the public parameters from the challenger in the $\mathbf{Exp}^{\mathsf{PKCH\text{-}Prf}}$ experiment. The reduction uses $P^{x_i^1}$ given from the challenger in experiment $\mathbf{Exp}^{\mathsf{PKCH\text{-}Prf}}$ to construct the public parameters. This is done by treating x_i^1 as $\hat{b}_i$ in the parameters, i.e., the reduction samples $b_i, \hat{v}_r, v_i \leftarrow\$ \mathbb{Z}_p$ and computes: $B_i \leftarrow P^{b_i}$, $\hat{B}_i \leftarrow \hat{X}_i^1$, $B_{\ell+i} = (X_i^1)^{b_i}$, $\hat{B}_{\ell+i} = (\hat{X}_i^1)^{b_i}$ $\forall i \in [\ell]$. The reduction then computes the verification bases as: $\forall i \in [\ell], \hat{V}_i = (X_i^1)^{\hat{v}_i}, \hat{V}_{i+\ell} = P^{\hat{v}_i}$. We can see that while the reduction does not receive any element like $P^{1/\hat{b}_i}$, the reduction can still create verification bases that are distributed identically to an honestly generated scheme as $\mathsf{dlog}_{\hat{V}_i}(\hat{V}_{i+\ell}) = 1/(x_i^1) = 1/(\hat{b}_i)$ exactly like Setup would generate normally. The reduction then computes the rest of the verification bases as follows: $\forall i \in [\ell], V_i = P^{v_i}, V_{i+\ell} = P^{v_i/b_i}$. This ensures that the verification bases, $\{V_i\}$, are computed correctly as well.

The reduction then generates its challenge key pair, $(\mathsf{sk}, \mathsf{pk})$, and gives the public key and parameters to the adversary.

The reduction then answers signature queries with the public parameters and challenge secret key.

The adversary then submits an adversarial public key and signature. The reduction then "presigns" the lower half of the message $\{M_i\}_{i \in [\ell]}$ with its challenge secret key (computing $M' = \{M_i^{x_i}\}$) and passes this message to the challenger in the proof experiment, $\mathbf{Exp}^{\mathsf{PKCH\text{-}Prf}}$. If the challenger's bit is $b = 0$, the challenger exponentiates this message M' with its secret key and forms a signature. We can see that because the reduction presigned the message with the reduction's secret key, this is distributed identically to a signature from the reduction's secret key as we treat the challenger's secret key as the $\hat{b}_i$ values in the parameters. If the challenger's bit, b, is 1, then this signature appears to be from a random secret key. To ensure the returned public key from the proof challenger verifies, we exponentiate it with b_i and concatenate the resulting ℓ dimension vector to the end of the returned public key.

We then have a second reduction that operates exactly like the last, but instead of using the reduction's secret key to presign the message to query to the challenger, it uses the adversary's extracted secret key. Our reduction must extract the secret key from the adversary. To do this, we leverage Lemma 35 to show that if the public key verifies, then we can extract the secret key in the GGM (thus ensuring that the extractor $\mathcal{E}_{\mathsf{pk}}$ is efficient).

We formalize this reduction in Fig. 14. This reduction is split into the two algorithms necessary for the $\mathsf{Exp}^{\mathsf{PKCH\text{-}Prf}}$ experiment and takes in a generic hybrid distinguisher, $\mathcal{A}$, along with the input from the experiment. We use $i = 1$ to denote the reduction that distinguishes between hybrids 1 and 2 and $i = 2$ to denote the reduction that distinguishes between hybrids 2 and 3. We can see that pk' is a valid public key because $(\hat{X}'_i)^{b_i} = \hat{X}_{i+\ell}$, which verifies when substituted into the verifying pairing equation: $e(V_i, \hat{X}_i) = e(V_{\ell+i}, \hat{X}_{\ell+i})$. Because we used the lower half of the message to pass to the PKCH-Prf challenger after exponentiating it with sk, this message has the form $\mathbf{M}' = \{{B_i}^{x_i}\}_{i\in[\ell]}$. If the PKCH-Prf's secret bit is $b = 0$, then they exponentiate with their secret key and form the signature $Z, Y, \hat{Y}$. Because we've included the challengers public key elements in our public parameters, the challenger's secret key is effectively equal to $\{\hat{b}_i\}_{i\in[\ell]}$ for our reduction's public parameters. Thus, the computation of Z will be: $(\prod({B_i}^{x_i})^{\hat{b}_i})^y = (\prod({B_{\ell+i}}^{x_i}))^y$, which is distributed identically to a signature from the reduction's secret key. If $b = 1$, this Z value will instead be: $(\prod({B_i}^{x_i})^{x_{2i}})^y$. This key effectively looks random to the hybrid distinguishing adversary, thus appearing as the second hybrid. While this signature and public key may not appear to verify as it does not depend on $\hat{b}_i$, we see that if this signature verifies in the [CL19] scheme, it will also verify in our scheme as the construction in [CL19] implies that $e(Z, \hat{Y}) = \prod_{i\in[\ell]} e(M'_i, \hat{X}_i) = 1$. We see that our verification is instead $e(Z, \hat{Y}) = \prod_{i\in[\ell]} e(M_i, \hat{X}'_i) = 1$ where $\hat{X}'_i = \hat{X}_i^{x_i}$ and $M'_i = M_i^{x_i}$ (or $\hat{X}'_i = \hat{X}_i^{x_{i,\mathcal{A}}}$ and $M'_i = M_i^{x_{i,\mathcal{A}}}$ if $i = 2$). Thus, because of the properties of bilinear pairings, the verification of the signature holds in our scheme and thus the adversary's view is identical to the real game.

Fig. 13. *Hybrid*$_i$

Hybrid$_i(1^\lambda)$
1: $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
2: $(\mathsf{sk}, \mathsf{pk}) = \mathsf{KGen}(\mathsf{pp})$
3: $(\mathsf{pk}^{\mathcal{A}}, \sigma^{\mathcal{A}}, \mathbf{M}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}_1(\mathsf{pp}, \mathsf{pk})$
4: $\mathsf{sk}^{\mathcal{A}} \leftarrow \mathcal{E}^{\mathcal{A}}_{\mathsf{sk}}(\mathsf{pk}_{\mathcal{A}})$
5: $\rho \leftarrow\$ \mathbb{Z}_p$
6: **if** $i = 0$,
7: $\quad \mathsf{pk}^\dagger = \mathsf{ConvertPK}(\mathsf{pk}, \rho)$
8: $\quad \sigma = \mathsf{Sign}(\mathsf{ConvertSK}(\mathsf{sk}, \rho), \mathbf{M})$
9: **if** $i = 1$,
10: $\quad (\mathsf{sk}^\dagger, \mathsf{pk}^\dagger) = \mathsf{KGen}(\mathsf{pp})$
11: $\quad \sigma = \mathsf{Sign}(\mathsf{sk}^\dagger, \mathbf{M})$
12: **if** $i = 2$,
13: $\quad \mathsf{pk}^\dagger = \mathsf{ConvertPK}(\mathsf{pk}^{\mathcal{A}}, \rho)$
14: $\quad \sigma = \mathsf{Sign}(\mathsf{ConvertPK}(\mathsf{sk}^{\mathcal{A}}, \rho), \mathbf{M})$
15: $b' \leftarrow \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{pk}^\dagger, \sigma)$
16: **return** b'

Thus, because the first hybrid is identical to the APKCH game when $b = 0$ and the third hybrid is indistinguishable from the APKCH game when $b = 1$, we've proven that these two cases are indistinguishable. □

B.2 Proofs of the extendable construction in Fig. 9

To show that any property of a non-extended scheme still holds for an extended scheme, observe that ExtendSetup will exponentiate each element in the public parameters with random scalars and that the resulting public parameters have the same distribution as the extended ones. Thus, our adversary cannot distinguish between these two cases (where ExtendSetup is called and when Setup is called). Thus, a reduction to any game on our original construction can be reduced by our extended

Fig. 14. APKCH proof reduction

$\mathcal{R}_1^{i,\mathcal{A}}(\mathbf{B^l}, \mathbf{B^u}, \hat{\mathbf{B}}^{\mathbf{l}}, \hat{\mathbf{E}}^{\mathbf{b}}) \to (\mathsf{st}, \mathbf{M}')$

$\mathbf{d} = \{\hat{d}_i\}_{i\in[\ell]} \leftarrow\$ \mathbb{Z}_p$

$\mathbf{r} = \{\hat{v}_i\}_{i\in[\ell]} \leftarrow\$ \mathbb{Z}_p$

$\mathbf{o} = \{v_i\}_{i\in[\ell]} \leftarrow\$ \mathbb{Z}_p$

$\mathsf{pp} = \{D_i, B_i, B_{\ell+i}, \hat{B}_i, \hat{B}_{\ell+i}, V_i, V_{i+\ell}, \hat{V}_i, \hat{V}_{i+\ell}\}_{i\in[\ell]}$ where $B_i, B_{\ell+i}, \hat{B}_i$ are drawn from the reduction's input, and $D_i = P^{\hat{d}_i}$, $\hat{B}_{\ell+i} = \hat{B}_i^{\hat{d}_i} =$, $V_i = P^{v_i\hat{d}_i}$, $V_{i+\ell} = P^{v_i\hat{d}_i}$, $\hat{V}_i = \hat{B}_i^{v_i}$, and $\hat{V}_{i+\ell} = P^{v_i}$

$(\mathsf{sk}, \mathsf{pk}) = \mathsf{KGen}(\mathsf{pp})$ such that $\mathsf{sk} = \{x_i\}_{i\in[\ell]}$ and $\mathsf{pk} = \{\hat{B}_i^{x_i}\}_{i\in[\ell]} || \{\hat{B}_{\ell+i}^{x_i}\}_{i\in[\ell]}$

$(\mathsf{pk}^{\mathcal{A}}, \sigma^{\mathcal{A}}, \mathbf{M}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}_1(\mathsf{pp}, \mathsf{pk})$

$\mathsf{sk}^{\mathcal{A}} \leftarrow \mathcal{E}_{\mathsf{sk}}^{\mathcal{A}}(\mathsf{pk}_{\mathcal{A}})$

if $i = 1, \mathbf{M}' = \{M_i^{x_i}\}_{i\in[\ell]}$

if $i = 2, \mathbf{M}' = \{M_i^{x_i^{\mathcal{A}}}\}_{i\in[\ell]}$

return $(\mathsf{st}, \mathbf{M}')$

$\mathcal{R}_2^{i,\mathcal{A}}(\mathsf{st}, \sigma) \to b$

if $i = 1, \mathsf{pk}' = \{(\hat{E}_i^b)^{x_i}\}_{i\in[\ell]}$

if $i = 2, \mathsf{pk}' = \{(\hat{E}_i^b)^{x_i^{\mathcal{A}}}\}_{i\in[\ell]}$

$\mathsf{pk}^\dagger = \mathsf{pk}' || (\mathsf{pk}')^{\hat{d}_i}$

$b' \leftarrow \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{pk}^\dagger, \sigma)$

return b'

construction by simply a reduction that simply ignores the pp' given by an adversary and instead using the pp from the challenger. This works because the ExtendSetup scheme effectively chooses a random set of valid public parameters and thus the one from the challenger is just as likely as any other set of public parameters. This works because the structure of public parameters can be verified by performing the following checks: $\forall i \in [\ell], e(V_i, \hat{B}_i) = e(V_{\ell+i}, \hat{B}_{\ell+i})$, $e(B_i, \hat{V}_i) = e(B_{\ell+i}, \hat{V}_{\ell+i})$ and $e(B_i, \hat{B}_i) = e(B_{\ell+i}, \hat{P})$. These checks ensure that the public parameters are structured correctly.

When calling FinalizeSetup, for any of the games, we can create a reduction that simply uses td to exponentiate elements from the challenger with b_i and $\hat{b}_i$ or exponentiate elements with $1/b_i$ and $1/\hat{b}_i$ from the adversary. This poses a non-trivial challenge as the adversary submits a signature which already has these $\hat{b}_i$ or b_i values in it which our reduction cannot do a pair-wise exponentiation to fix. But fortunately, if we simply update the verification bases of the public key with $1/\hat{b}_i$ or $1/b_i$, we can see that it verified with the unfinalized public parameters and the rest of the public key, signature, and message are distributed correctly. This makes our reduction valid for the unforgeability game, but in the APKCH game, the challenger will then return a randomized public key and signature. We can see that after receiving the public key from the APKCH challenger, this reduction can simply exponentiate it with b_i and $\hat{b}_i$ to make it look right for the adversary and thus, our reduction works for any of the games.

Thus, because we can replace the output of ExtendSetup with fresh parameters, and also, we can translate messages and public keys between the schemes after using FinalizeSetup to win the security games with our reduction, our proofs hold when using $\mathsf{Setup}^{\mathsf{ext}}$ instead of Setup □

B.3 DAC proof

Proof of Thm. 21 (Correctness). **Root issuance:** Because our signatures verify when honestly created, we can see that a credential issued from the root will verify. Because our mercurial signatures are correct, this is true even after the signature and message (delegator key) have been randomized. **Delegator issuance:** We can see that if cred_1 verifies, then cred_2 will contain that chain, which still verifies even when randomized. Because the key space for level L^* is the message space for L' (with equivalence classes matching as well) when we randomize the key for level L', it matches the message space of the last signature from cred_1 and thus verifies due to correctness (similarly for keys/signatures for all $L^\dagger < L'$).

Proof of Thm. 22 (Unforgeability). From the game in Def. 19, we know that if the adversary wins with non-negligible probability, then one of three cases occurred, either (1) $\mathsf{sk}_0 \neq \mathsf{sk}_{rt}$, (2) $\forall i \in [L'], (\mathsf{sk}_i, i) \notin$

$\mathsf{DEL}_{\mathcal{A}}$, or (3) $\exists i \in [L']$, s.t. $\mathsf{sk}_i \in \mathcal{SK}_{DL}$. We'll start with the first case, where $\mathsf{sk}_0 \neq \mathsf{sk}_{rt}$. We can see that our construction checks to ensure that the first public key is the root's key so this cannot occur. In the second case, we have that $\forall i \in [L'], (\mathsf{sk}_i, i) \notin \mathsf{DEL}_{\mathcal{A}}$. In this case, we have that no key in the credential chain was actually delegated to that adversary. In this case, we can either reduce to the unforgeability of the scheme, or recover a secret key of an honest user through the adversary's proof of knowledge of the end user. In the third case, we have that $\exists i \in [L']$, s.t. $\mathsf{sk}_i \in \mathcal{SK}_{DL}$. Because the TRA reveals the secret keys of any blacklisted user, and recognizing public keys succeeds with probability 1, we see that this cannot happen. Thus, our DAC scheme is unforgeable.

Proof of Thm. 23 (Anonymity). We can see in the anonymity game that the only difference is between $b = 0$ and $b = 1$ is when the adversary receives the anonymity challenge. We will construct hybrids that replace the non-root public keys with random public keys starting from the lowest level and working up. We can create hybrids that replace public keys and signatures from the bottom up to prove that either credential that the challenger picks is indistinguishable from a random chain.

Definition 38 ($Hybrid_{b,j}$).

1: $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^L)$
2: $(tsk, tpk) \leftarrow \mathsf{Setup}(\mathsf{pp})$
3: $(st, \mathsf{pk}_{rt}, \mathsf{sk}_0, \mathsf{cred}_0, \mathsf{sk}_1, \mathsf{cred}_1, L') \leftarrow \mathcal{A}_0^{\mathcal{O}^{\mathsf{RegisterUser}}(\cdot), \mathcal{O}^{\mathsf{RevokeUser}}(\cdot)}(\mathsf{pp})$
4: $\forall i \in \{0,1\}$, **if** $(\mathsf{Prove}(\mathsf{pp}, \mathsf{sk}_i, \mathsf{cred}_i, L') \leftrightarrow \mathsf{Verify}(\mathsf{pk}_{rt}, L', DL)) \neq 1$, **return** $\perp$
5: $\{(\mathsf{pk}_i, \sigma_i, tok_i)\} \leftarrow \mathsf{cred}_b$
6: $\mathsf{sk}_{j-1} = \mathcal{E}_{\mathsf{pk}}(\mathsf{pk}_{j-1})$
7: $\forall i \in [L' - j]$,
8: $\quad (\mathsf{sk}_{j+i-1}, \mathsf{pk}^*_{j+i-1}) \leftarrow \mathcal{PK}(\mathsf{pp})$,
9: $\quad tok^*_{j+i-1} \leftarrow \mathsf{RegisterUser}(\mathsf{pp}, tsk, \mathsf{pk}_{j+i-1})$
10: $\quad \sigma^*_{j+i-1} \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}_{j+i-2}, \mathsf{pk}_{j+i-1})$
11: $\mathsf{cred}^* = \{(\mathsf{pk}_i, \sigma_i, tok_i)\}_{i \in [j]} \| \{(\mathsf{pk}^*_{j+i-1}, \sigma^*_{j+i-1}, tok^*_{j+i-1})\}$
12: $\mathsf{Prove}(\mathsf{pp}, \mathsf{sk}^*_{L'}, \mathsf{cred}^*, L') \leftrightarrow \mathcal{A}_1(st) \rightarrow b'$

To create these hybrids, the challenger in these hybrids will extract the secret keys from the public keys in the chains. We construct our reduction from an adversary that can distinguish hybrid j and $j + 1$ to APKCH in Def. 39.

After we've proven that $Hybrid_{0,0}$ is indistuiguishable from $Hybrid_{0,L}$, we can again create hybrids that instead use the public keys in cred_1 in the anonymity challenge, thus proving that $Hybrid_{1,0}$ is indistinguishable from $Hybrid_{1,L}$. We then observe that $Hybrid_{0,L}$ is identical to $Hybrid_{1,L}$ as all keys have been replaced with random ones. Thus, at either end of our hybrids, we'll have either case of the anonymity challenge and thus prove that they are indistuighuishable.

Definition 39 (Reduction of distinguishing $Hybrid_{b,j}$ from $Hybrid_{b,j+1}$ to APKCH).

$\mathcal{R}_1^{\mathcal{O}^{\mathsf{Sign}}(\mathsf{sk}, \cdot)}(\mathsf{pk}, \mathsf{pp}) \rightarrow (\mathsf{pk}^*, \sigma^*, M^*, \mathsf{st})$

1: *If $j = L - 1$, the reduction interacts with* $\mathsf{Setup}^{\mathsf{ext}}$ *in the* APKCH *game by first supplying $\perp$ (as* pp_0 *on line Line 1 of* $\mathsf{Setup}^{\mathsf{ext}}$ *in Fig. 7) to make the challenger generate the parameters using* Setup. *The reduction receives the public parameters from the challenger which they label* pp'_L *(these parameters are generated by the challenger on line Line 3 of Fig. 7 in* $\mathsf{Setup}^{\mathsf{ext}}$ *and in that function they are labeled* pp_1*). The reduction then generates* $(\mathsf{pp}'_{L-1}, td'_{L-1}) \leftarrow \mathsf{ExtendSetup}(\mathsf{pp}'_L)$ *and returns td_{L-1} to the challenger (on line Line 4 of* $\mathsf{Setup}^{\mathsf{ext}}$ *as td') which allows the challenger to finalize their setup parameters. The challenger then returns the finalized parameters, which the reduction labels* pp_L*. The reduction then generates* $\forall i \in [L-1](\mathsf{pp}'_{i-1}, td'_{i-1}) \leftarrow \mathsf{ExtendSetup}(\mathsf{pp}'_i)$ *and* $\forall i \in [L-1] \setminus \{L\}, (\mathsf{pp}_i, td_i) \leftarrow \mathsf{FinalizeSetup}(\mathsf{pp}'_i, td'_i, td'_{i-1})$*. The reduction then computes* $\mathsf{pp}_{\mathsf{DAC}} = \{\mathsf{pp}_i\}_{i \in [L]}$.

2: *If $j \neq L - 1$, the reduction generates* $(\mathsf{pp}'_L, td'_L) = \mathsf{Setup}(1^\lambda, 1^{\ell=2})$ *and then calls* $\forall i \in [L - j], (\mathsf{pp}'_{j+i}, td'_{j+i}) \leftarrow \mathsf{ExtendSetup}(\mathsf{pp}_{j+i+1})$*. The reduction then interacts with* $\mathsf{Setup}^{\mathsf{ext}}$ *in the* APKCH *game by supplying* pp'_{j+1} *as the parameters labeled* pp_0 *in* $\mathsf{Setup}^{\mathsf{ext}}$ *in Fig. 7. The reduction labels these parameters as* pp'_j*. The challenger then extends this computing* $(\mathsf{pp}'_{j-1}, td'_{j-1}) = \mathsf{ExtendSetup}(\mathsf{pp}'_j)$ *and returns* td'_{j-1} *to the challenger (labeled td' in Fig. 7). The challenger then*

returns public parameters which the reduction labels as pp_j. *The reduction then finishes initializing the scheme by computing* $\forall i \in [j-2], (\mathsf{pp}'_i, td'_i \leftarrow \mathsf{ExtendSetup}(\mathsf{pp}'_{i+1})$. *The reduction then calls* $\forall i \in [j-1], (\mathsf{pp}_i, td_i) \leftarrow \mathsf{FinalizeSetup}(\mathsf{pp}'_i, td'_i, td'_{i-1})$ *to finish the parameters,* $\mathsf{pp}_{\mathsf{DAC}} = \{\mathsf{pp}_i\}_{i\in[L]}$.

3: $(tsk, tpk) \leftarrow \mathsf{TKeyGen}(\mathsf{pp}_{\mathsf{DAC}})$

4: $(st, \mathsf{pk}_{rt}, \mathsf{sk}_0, \mathsf{cred}_0, \mathsf{sk}_1, \mathsf{cred}_1, L') \leftarrow \mathcal{A}_0^{\mathcal{O}^{\mathsf{RegisterUser}}(\cdot),\mathcal{O}^{\mathsf{RevokeUser}}(\cdot)}(\mathsf{pp}_{\mathsf{DAC}})$

5: $\forall i \in \{0,1\}$, **if** $(\mathsf{Prove}(\mathsf{pp}, \mathsf{sk}_i, \mathsf{cred}_i, L') \leftrightarrow \mathsf{Verify}(\mathsf{pk}_{rt}, L', tpk)) \neq (1, *)$, **return** $\perp$

6: $\{(\mathsf{pk}_i, \sigma_i, tok_i)\} \leftarrow \mathsf{cred}_b$

In the next line, the reduction returns their public key pk_j, *signature* σ_j, *and message,* pk_{j+1} *to the* APKCH *challenger.*

7: **return** $(\mathsf{pk}_j, \sigma_j, \mathsf{pk}_{j+1}, st = (tsk, \mathsf{pp}_{\mathsf{DAC}}))$

$\mathcal{R}_2^{\mathcal{O}^{\mathsf{Sign}}(\mathsf{sk},\cdot)}(\mathsf{pp}, \mathsf{st}, \mathsf{pk}^{b^*}, \sigma^{b^*}) \rightarrow b'$

Here, the reduction has received $\mathsf{pk}^{b^*}, \sigma^{b^*}$ *from the* APKCH *challenger which is either a randomization of* pk_j, σ_j *or a new signature form an unrelated key.*

8: $tok^* \leftarrow \mathsf{RegisterUser}(\mathsf{pp}, tsk, \mathsf{pk}^{b^*})$

9: $\mathsf{cred}^* = \{(\mathsf{pk}_i, \sigma_i, tok_i)\}_{i\in[j]}$
$\|\{(\mathsf{pk}^{b^*}, \sigma^{b^*}, tok^*)\}$
$\|\{(\mathsf{pk}_{j+i}, \sigma_{j+i}, tok_{j+i})\}_{i\in[L'-j]}$

10: $\mathsf{Prove}(\mathsf{pp}_{\mathsf{DAC}}, \mathsf{sk}^*_{L'}, \mathsf{cred}^*, L') \leftrightarrow \mathcal{A}_1(st) \rightarrow b'$

Analysis. We can see that if the APKCH challenger's bit, b, is 0, this appears identical to $Hybrid_{b,j}$. This is because the challenger returns a pk_b in the same equivalence class as pk_j. Because our scheme achieves perfect origin-hiding, this means that this appears exactly as if a real user had randomized it. If b is 1, then pk_b is a random public key, and we appear exactly as $Hybrid_{b,j+1}$. This is because the pk_b returned is now a random public key.

For issuing, we can see that the proof is similar, except that the issuer produces a credential chain on a message from the adversary as long as that message verifies.