Non-Interactive Threshold Mercurial Signatures with Applications to Threshold DAC

Scott Griffy[1], Nicholas Jankovic[1], Anna Lysyanskaya[1], and Arup Mondal[2]

[1] Brown University
scott_griffy@brown.edu, nicholas_jankovic@brown.edu,
anna_lysyanskaya@brown.edu
[2] Ashoka University
arup24mondal@gmail.com

Abstract. In a mercurial signature, a signer signs a representative m of an equivalence class of messages on behalf of a representative pk of an equivalence class of public keys, receiving the signature σ. One can then transform σ into a signature σ' on an equivalent (to m) message m' under an equivalent (to pk) public key pk'. Mercurial signatures are helpful in constructing delegatable anonymous credentials (DACs): their privacy properties enable straightforward randomization of a credential chain, hiding the identity of each signer while preserving the authenticity of the overall credential.

Unfortunately, without trusted setup, known constructions of mercurial signatures satisfy only a weak form of this privacy property. Specifically, an adversary who is responsible for a link in a delegation chain—and thus knows its corresponding secret key—will be able to recognize this link even after the chain has been randomized.

To address this issue, Abe et al. (Asiacrypt 2024) proposed (interactive) *threshold mercurial signatures* (TMS), which remove the reliance on a single trusted signer by distributing the signing capability among multiple parties, none of whom knows the signing key. However, this contribution was far from practical, as it required the signers to interact with each other during the signing process.

In this work, we define and construct the first *non-interactive* TMS, where each signer computes its partial signature locally, and any t of them combine into an ordinary mercurial signature. This is done using the mercurial signature of Mir et al. (CCS 2023) as a starting point. We also enable the Mir et al. construction to sign its own public keys, a property that is essential for DACs but had been left as an open question. Taking these two contributions together, we build the first non-interactive *threshold delegatable anonymous credentials* with non-transferable credentials.

1 Introduction

Services that preserve user anonymity are becoming increasingly important in the digital era. Without them, ordinary people risk being surveilled by powerful

entities such as governments, corporations, or hackers. One way to provide privacy while maintaining service integrity (*i.e.,* authorizing only certain users) is through *anonymous credentials*. They were first described by Chaum [18], first realized by Camenisch and Lysyanskaya [15], and then improved and extended in many subsequent works [8, 13, 14, 17, 20, 30, 40]. Anonymous credentials allow a user to prove in zero-knowledge that they hold a credential, *i.e.*, an issuer's signature on their attributes or public key, without revealing it. They seem to be the perfect answer to legislative efforts towards privacy-preserving digital identity, such as the EU's Digital Identity Wallet regulation (EUDI) [27].

For example, every EU citizen should be able to prove citizenship without revealing their member state. This can easily be achieved if the EU issues all credentials; but in practice, it is much more feasible for issuance to be carried out by member states and regional authorities below them. This is resolved by *delegatable anonymous credentials* (DACs), introduced by Chase and Lysyanskaya [17]. DACs allow a root issuer to delegate issuing power to multiple delegatees, each of whom can then issue credentials to users without further interaction with the root. Importantly, DACs ensure that intermediary issuers (e.g. the member state) are not revealed in a credential showing.

DACs can be realized efficiently and modularly with *mercurial signatures*, a scheme introduced by Crites and Lysyanskaya [20] that allows keys, messages, and signatures to be randomized. Using this in the context of DAC lets the public keys of delegated signers be randomized, thereby hiding the identity of the intermediate signer while still proving the user holds a credential from a trusted root issuer. Since their introduction, mercurial signatures have attracted significant research attention [4, 21, 35, 40].

Threshold signatures [23,24] are another powerful tool for credential schemes, as they allow the distribution of authority between signers. The notion of threshold privacy-preserving credentials has been explored in a number of works [26,45]. Thresholdizing a mercurial signature can strengthen both anonymity and unforgeability [4]. In many constructions, a single issuer holding sk recognizes its randomized key pk' when a credential holder reveals it for verification. But when sk is shared, no coalition below the threshold can recognize pk'.

This motivates the need for a *threshold mercurial signature* (TMS) scheme to provide anonymous credentials with distributed signing power. Thresholdized mercurial signatures were first introduced in [4] and used to construct mix-nets in [3]. However, all existing constructions of TMS require interaction between the signers to produce a signature. This raises the following question:

Can we design a non-interactive threshold mercurial signature scheme?

We answer this question in the affirmative by formally defining and constructing the first such scheme. Then, for a mercurial signature to be used in the construction of a DAC scheme, its public keys must be signable as messages. This motivates the following question:

Can such a threshold mercurial signature scheme sign its own public keys?

We answer this question in the affirmative by modifying the public key space and proving the security of this modification. This enables us to extend our non-interactive TMS and realize the first *threshold delegatable anonymous credentials* (TDAC) scheme.

1.1 Our Contributions

In this work, we study efficient constructions of threshold mercurial signatures and their application to threshold DAC. Our main contributions are as follows:

- We construct the first *non-interactive* threshold mercurial signature scheme. Our construction builds on the mercurial signature of Mir et al. [40] which is itself based on the structure-preserving signature of Ghadafi [33].
- We further extend our construction to support the signing of public keys, solving an open problem left by [40]. This extension requires revealing additional information about the signing key; we prove in the generic group model that this does not give an adversary any advantage. Thus, we overcome a key limitation from the scheme in [40] that prevented its use in delegatable anonymous credentials (DAC).
- We achieve the first non-interactive, non-transferable threshold DAC scheme from mercurial signatures. Threshold DAC was previously studied by Mir, Slamanig, and Mayrhofer [41], but their construction requires an extra round during showing (due to commitments needed for proving correct encryption). Moreover, their scheme [41] does not enforce non-transferability, which is the property that prevents users from maliciously sharing credentials. Non-transferability is a requirement for anonymous credentials [15]. In general, it is achieved by signing users' public keys, enabling users to prove their identity. Since Mir et al. [41] supports only attribute signing, their construction cannot bind credentials to public keys, and thus cannot prevent malicious credential transfer. In contrast, our construction incorporates users' secret keys into the signed messages, thereby achieving non-transferability.

In Table 1, we compare our mercurial signature to related work. The closest work to ours is Mir et al. [40], and we highlight the differences. In particular, our construction supports thresholdization and DAC while theirs does not. We can estimate the computational cost of our scheme with zk-bench [28]. The pairings in signature verification will take up the majority of the computation as computing our signatures always takes two multi-scalar multiplications along with computing a hash function. Verifying a signature when $\ell = 2$ (the smallest possible scheme) will take 11 pairings, which take 19ms with BLS12-381 using the zkcrypto library on AWS EC2 m5.large according to zk-bench. Verifying a TDAC credential with 3 levels requires another 23 pairings, which take 41ms in the same environment. Thus, in total, verifying a chain of signatures of length 3 will take 60ms.

These costs are borne by the verifier: the holder computes no pairings when showing a credential, since Prove consists only of exponentiations in $\mathbb{G}_1$ and

$\mathbb{G}_2$ and a Schnorr-style proof of knowledge, as in BBS-based proposals for the EUDIW [7]. The holder computes pairings only once per credential, at issuance (in Reconst).

Schemes	Crites et al. [20]	Griffy et al. [35]	Abe et al. [4]	Mir et al. [40]	Ours (Figure 4)
$\lvert\mathsf{pp}\rvert$	$O(1^\lambda)$	$4\ell(\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_2\rvert)$	$O(1^\lambda)$	$O(1^\lambda)$	$O(1^\lambda)$
$\lvert\mathsf{pk}\rvert$	$\ell \cdot (\lvert\mathbb{G}_2\rvert)$	$2\ell \cdot (\lvert\mathbb{G}_2\rvert)$	$\ell \cdot (\lvert\mathbb{G}_2\rvert)$	$\ell \cdot (\lvert\mathbb{G}_2\rvert)$	$\ell \cdot (\lvert\mathbb{G}_2\rvert + 2\lvert\mathbb{G}_1\rvert)^{\dagger\dagger}$
$\lvert\sigma\rvert$	$3 \cdot \lvert\mathbb{G}_1\rvert$	$3 \cdot \lvert\mathbb{G}_1\rvert$	$3 \cdot \lvert\mathbb{G}_1\rvert$	$3 \cdot \lvert\mathbb{G}_1\rvert$	$3 \cdot \lvert\mathbb{G}_1\rvert$
$O(\mathsf{Verify})$	$(\ell + 3) \cdot e$	$(5\ell + 3) \cdot e$	$(\ell + 3) \cdot e$	$(4\ell + 3) \cdot e$	$(4\ell + 3) \cdot e$
Threshold	×	×	✓	×†	✓
Non-interactive	N/A‡	N/A‡	×	✓	✓
Supports DAC	✓	✓	✓	×	✓

Table 1: Concrete parameter comparison (in terms of $\mathbb{G}$ elements).

ℓ denotes the length of signable messages. "Non-interactive" means that issuers do not need to interact with each other for a signature to be combined. A non-constant $\lvert\mathsf{pp}\rvert$ (beyond the security parameter) implies trusted setup (any non-trusted setup can be sampled in the ROM). $O(\mathsf{Verify})$ is the complexity of signature verification and $X \cdot e$ indicates that X pairings must be computed.

† This scheme supports aggregation, but not t-out-of-n thresholdization.

‡ It is trivial for schemes where signatures are not thresholdized or aggregated to be non-interactive.

†† The extra key size allows for DAC to be constructed from this signature. For only thresholdization, the key size is the same as [40].

The security of our construction relies on the generic group model (GGM) [44] and random oracle model (ROM) for the hash function H, as does the construction of [40] on which we build. Our own proofs use these models to prove unforgeability, for which our notion differs from that of [40], and to show that the elements revealed when a public key is signed do not help an adversary. Thresholdization is proven by a standard-model reduction to the underlying non-threshold scheme (that of [40], or our extension of it); since this reduction is generic, the GGM security of the underlying scheme carries over to our threshold scheme.

This reliance on idealized models is in line with prior work. Mercurial signatures are known to require interactive assumptions [6], and existing constructions rely on generic or algebraic group models in their security proofs [4,20,21,35,40]. Idealized models weaker than the GGM offer little here. Even in the algebraic group model, the public key class-hiding adversary outputs a bit rather than a group element, so there is no algebraic representation to exploit.

Limitations. Supporting DAC increases the size of public keys in our scheme when compared to [40]. As we extend our scheme to the delegatable setting, our key sizes double with each added delegation level. This is acceptable as long as the number of delegations is small. For example, a real-world application might use credential chains of length 3. In this case, the largest public key in our scheme takes 22 group elements, or about 1000 bytes. This is still much more efficient than DAC schemes from Groth-Sahai proofs [8].

A second limitation arises due to our use of tag-based Diffie-Hellman message spaces. Similarly to [40], the signer must recompute the hash to ensure unforgeability, which reveals the user's identity. Fortunately, this only affects the privacy of the user during issuing, where the signer often needs to know the user's identity to verify them anyway. Showings still do not reveal any unnecessary information about the user and are unlinkable to any issuing.

Third, the privacy gained in our scheme—the fact that even signers cannot recognize their own signature when randomized—assumes a trusted dealer of keys or the use of a distributed key generation (DKG) protocol. While we do not provide a dedicated DKG construction for our scheme, existing DKG protocols for discrete log-based threshold signatures (e.g., [25, 32, 38]) suggest a natural pathway to jointly generate our construction's secret key shares. Detailed instantiation and efficiency analysis is left for future work.

Finally, our credentials certify keys, not attributes. Attribute-level selective disclosure can be layered on with randomizable set commitments as in [40] (which requires a trusted commitment key); we leave this orthogonal extension out.

1.2 Technical Overview

Our threshold mercurial signature takes the approach of Crites et al. [19] to thresholdize the secret key using Shamir secret sharing [43]. This approach relies on certain homomorphic properties of the underlying signature scheme. Specifically, for two valid keys-signatures pairs (pk, σ) and (pk', σ'), the product $\sigma * \sigma'$ is a valid signature under the public key $\mathsf{pk} * \mathsf{pk}'$. Such *key-homomorphic signatures* are studied in depth by [22].

Our scheme is based on the key-homomorphic signature of [40], over bilinear groups $\mathbb{G}_1$ and $\mathbb{G}_2$ generated by P and $\hat{P}$. A message scalar $\vec{m}$ is encoded as the pair $(\vec{M}, \vec{N}) \in \mathbb{G}_1^2 \times \mathbb{G}_2^2$, where $N_j = \hat{P}^{m_j}$ and $M_j = T_j^{m_j}$, relative to a tag $\vec{T} = (T_1, T_2)$. Here $T_j = h^{\rho_j}$, where h is a hash of $(P^{\rho_j})_j$, $\vec{N}$, and the signer's public key. A message is signed under $\mathsf{sk} = (x, y_1, y_2, z_1, z_2)$ as $\sigma = (h, b, s) = (h, T_1^{z_1} T_2^{z_2}, h^x M_1^{y_1} M_2^{y_2})$. The last two components are key-homomorphic: for signatures (h, b, s) and (h, b', s') on the same message and tag under sk and sk', $b \cdot b' = T_1^{z_1 + z_1'} T_2^{z_2 + z_2'}$ and $s \cdot s' = h^{x + x'} M_1^{y_1 + y_1'} M_2^{y_2 + y_2'}$ yield a signature under $\mathsf{sk} + \mathsf{sk}'$. We exploit this to construct threshold signatures. The key sk is shared coordinate-wise with Shamir secret sharing into shares $\mathsf{sk}_i = (x_i, y_{i,1}, y_{i,2}, z_{i,1}, z_{i,2})$, so that, *e.g.*, $\sum_{i \in \mathfrak{T}} \lambda_{i,\mathfrak{T}} x_i = x$ for every $\mathfrak{T} \subseteq [n]$ with $|\mathfrak{T}| = t$, where $\lambda_{i,\mathfrak{T}}$ is the Lagrange coefficient of party i. Since h depends only on the message, the tag secret, and the global public key, all signers derive the same h without interacting, and any t partial signatures combine into a signature that verifies under the global public key.

The intuition behind our second contribution (adapting [40] to support the signing on public keys) is more subtle. For public keys to be signable, they have to be adapted to fit the message space. The challenge is that, in the original scheme, messages span *both* source groups, while public keys lie only in the second source group: $\mathsf{pk} = (\hat{P}^x, \hat{P}^{y_1}, \hat{P}^{y_2}, \hat{P}^{z_1}, \hat{P}^{z_2})$. Thus, signing a public key

means extending it with extra elements in the first source group. We do this by mimicking the structure of messages: when pk is signed, its owner reveals a tag $\vec{T} = (h^{\rho_i})_{i\in[5]}$ and $\vec{M} = (T_i^{\mathsf{sk}_i})_{i\in[5]}$, where h is a hash of $(P^{\rho_i})_{i\in[5]}$, pk, and the public key that will sign pk.

Since this modification reveals new elements that are correlated with the secret key, it is not immediately clear that the scheme is still secure. Suppose, instead, that we had revealed the $\mathbb{G}_1$ terms $(P^x, P^{y_1}, P^{y_2}, P^{z_1}, P^{z_2})$. Then, $\sigma^* = (P, P^{z_1}P^{z_2}, P^xP^{y_1}P^{y_2})$ is a valid signature on the message $(\vec{M}^*, \vec{N}^*) = ((P, P), (\hat{P}, \hat{P}))$ with tag $\vec{T}^* = (P, P)$, which is a forgery. Fortunately, the modification we make is secure, which we prove in the generic group and random oracle models. For unforgeability, we give a direct proof, since we also adapt the notion of unforgeability to DAC (Section 3.1). For public key class-hiding, we reduce to the original scheme of [40]. The reduction cannot compute the extra $\mathbb{G}_1$ elements without the secret key, so we show that the adversary's queries cannot usefully depend on them.

1.3 Related Work

Anonymous Credentials. Delegatable anonymous credentials (DACs) were first realized by Chase and Lysyanskaya [17] but were not made efficient for long chains until the work of Belenkiy et al. [8], which relied on Groth-Sahai proofs [36]. Since Groth-Sahai proofs are often considered inefficient, Crites and Lysyanskaya [20] recently introduced the notion of mercurial signatures as a more practical foundation for constructing efficient DACs.

Threshold Signatures. Threshold signatures [23, 24] share the signing key among n parties so that any t of them can sign, while $t-1$ corrupted parties cannot forge; they are used, *e.g.*, in blockchains [31,39] and multi-factor authentication [29]. Threshold structure-preserving signatures were introduced in [19] and later achieved under standard assumptions with adaptive security [42]. Structure-preserving signatures lack the equivalence-class randomization that DAC requires, and mercurial signatures are known to require interactive assumptions [6], so these techniques do not directly transfer to our setting.

Mercurial Signatures. Mercurial signatures were introduced in [20] as a combination of equivalence classes on messages [30] and equivalence classes on public keys [5]. Mercurial signatures were initially used to realize more efficient DACs that out-performed the previous construction built using Groth-Sahai proofs [8], but have also been used in other applications, such as contact tracing [34]. Some mercurial schemes [20, 40] suffer from a limitation: signature unlinkability only holds when the signer does not collude with the adversary. Griffy et al. [35] strengthened the definition and construction of mercurial signatures from [20] to support privacy against malicious signers using a structured common reference string (SRS).

Many mercurial schemes are also *structure-preserving signatures* [1]. A structure-preserving signature has the property that its messages, signatures,

and keys are all composed of elements of a bilinear group. This enables the signature verification algorithm (and other functions) to be executed in zero knowledge using Groth-Sahai proofs [36]. Groth-Sahai proofs are *randomizable*, which is what allowed the efficient construction of DACs in [8].

Threshold Mercurial Signatures. Threshold mercurial signatures were first introduced by Abe et al. in [4], which adapted [20] to the threshold setting to create stronger anonymity and unforgeability properties for delegatable anonymous credentials. While their construction does not require an SRS (unlike [35]), their scheme does require interaction between signers to produce a signature, which is undesirable. Similar to threshold mercurial signatures is the work of Mir et al. [40] which introduced *aggregatable* mercurial signatures. However, their scheme did not support the signing of public keys as messages [34], which limited their application to issuer-hiding [12] and multi-authority [37] ACs (which are implied by DAC, but not vice-versa).

1.4 Disclosure of AI Use

AI provided limited contribution to this paper, in the sense of the IACR policy on generative AI.

2 Preliminaries

In this section, we briefly introduce the notation, definitions, and building blocks that we use in the rest of the paper. Mercurial signatures are presented alongside our threshold definitions in Section 3.

Notation. We use $\lambda \in \mathbb{N}$ to denote the computational security parameter. In threshold settings, we let n denote the number of parties and t denote threshold such that any fewer than t colluding parties cannot compromise security. Hence $1 \leq t \leq n$. We let $[b]$ denote the set $\{1, \ldots, b\}$. We denote the concatenation of x and y by $(x \| y)$. Given a set $\mathcal{X}$, we use $x \xleftarrow{\$} \mathcal{X}$ to denote the sampling of a value x from the uniform distribution over $\mathcal{X}$. We denote $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$ as any generic (unspecified) polynomial and negligible functions in λ, respectively. We abbreviate *probabilistic polynomial time* as PPT. For a turing machine TM and set of inputs args, we let $x \in \mathsf{TM}(\mathsf{args})$ mean that there exists random tape ρ such that $x = \mathsf{TM}(\mathsf{args}; \rho)$. We let $\mathsf{out} \leftarrow \mathcal{A}(\mathsf{in}; r)$ denote the evaluation of a PPT algorithm $\mathcal{A}$ that produces an output out from an input in with randomness $r \xleftarrow{\$} \{0,1\}^*$, and omitting r when it is obvious or not explicitly required. We let $\mathcal{A}^{\mathcal{O}^{\mathsf{alg}}}$ denote that we run $\mathcal{A}$ with oracle access to $\mathcal{O}^{\mathsf{alg}}$, *i.e.*, $\mathcal{O}$ executes alg on inputs of $\mathcal{A}$'s and returns the corresponding outputs. When invoking an adversary multiple times in one security game, the adversary is allowed to read and update their own state between invocations even if we do not explicitly state this.

Prime-Order Bilinear Groups. We work with cyclic groups of prime order p equipped with an asymmetric bilinear map. We assume a PPT bilinear group generator algorithm BGGen that, on input $\lambda \in \mathbb{N}$, outputs $\mathsf{BG} :=$

$(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, \hat{P}, e) \leftarrow \mathsf{BGGen}(1^\lambda)$. Here p is a prime of $\Theta(\lambda)$ bits; $\mathbb{G}_1, \mathbb{G}_2$ are bilinear groups (or pairing groups) and $\mathbb{G}_T$ is a multiplicative group, all of prime order p; P and $\hat{P}$ are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively; and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate, efficiently computable bilinear pairing. Hence, we see that $e(P, \hat{P})$ is a generator of $\mathbb{G}_T$.

Shamir Secret Sharing [43]. Shamir secret sharing lets a secret $s \in \mathbb{Z}_p$ be shared among n parties, such that any t can collaborate to reconstruct the original. $\mathsf{SSS.Share}(s, n, t)$ samples a uniformly random polynomial f of degree $t-1$ over $\mathbb{Z}_p$ with $f(0) = s$, and gives party P_i for $i \in [n]$ the share $s_i = f(i)$. Any $t-1$ shares are uniformly distributed and independent of s, making the scheme information-theoretically secure. Since f is determined by any t of its evaluations, for every $\mathfrak{T} \subseteq \{0, \ldots, n\}$ with $|\mathfrak{T}| = t$, and every point k,

$$f(k) = \sum_{i \in \mathfrak{T}} \lambda_{i,\mathfrak{T}}(k)\, f(i), \qquad \lambda_{i,\mathfrak{T}}(k) = \prod_{j \in \mathfrak{T} \setminus \{i\}} \frac{k-j}{i-j} \bmod p.$$

The secret s can therefore be reconstructed by computing $s = \sum_{i \in \mathfrak{T}} \lambda_{i,\mathfrak{T}}(0) s_i$ for any $\mathfrak{T} \subseteq [n]$ of size t. In our construction, we apply this interpolation in the exponent, and our reductions use it with $\mathfrak{T} = \mathcal{C} \cup \{0\}$ to derive honest parties' partial keys from the global key and the corrupted shares. We write $\mathsf{SSS.Share}(\vec{s}, n, t)$ to denote the coordinate-wise sharing of the vector $\vec{s}$, and $\lambda_{i,\mathfrak{T}}$ as shorthand for $\lambda_{i,\mathfrak{T}}(0)$.

Tag-based Message Spaces. Following [19,20,40], we first describe the space on which messages are signed before defining our mercurial signatures.

Tag-based Diffie-Hellman Message Space. Early constructions of DACs restricted the message space of the signature scheme to vectors of elements from a single source group (*e.g.*, $\vec{M} \in \mathbb{G}_1^\ell$). This restriction enabled the use of Groth-Sahai proofs [36] and allowed for efficient DAC constructions [8]. However, it was later shown that using only a single source group leads to impossibility results concerning signatures size [2]. This limitation was mitigated by defining message spaces over multiple source groups [33] (*e.g.*, $(\vec{M}, \vec{N}) \in \mathbb{G}_1^\ell \times \mathbb{G}_2^\ell$), which are referred to as *Diffie-Hellman (DH) message spaces* [19].

Why tags. A message must be randomizable within its equivalence class, in a way that cannot be linked to the original. In a DH message space, this rules out the natural encoding of a message secret $\vec{m}$ as $(\vec{M}, \vec{N})$ with $M_j = P^{m_j}$ and $N_j = \hat{P}^{m_j}$, randomized to $(\vec{M}^\nu, \vec{N}^\nu)$. Since the pairing spans both groups, anyone can link $(\vec{M}, \vec{N})$ to a randomization $(\vec{M}', \vec{N}')$ by checking whether $e(M_1, N_2') = e(M_2, N_1')$ [40]. Tags break this link. Each message comes with a tag $\vec{T} = (T_1, \ldots, T_\ell) = (h^{\rho_1}, \ldots, h^{\rho_\ell})$ for a group element h and a secret $\vec{\rho}$, and the $\mathbb{G}_1$-part becomes $M_j = T_j^{m_j}$. The two sides of the test above are now $e(h, \hat{P})^{\rho_1 m_1 m_2 \nu}$ and $e(h, \hat{P})^{\rho_2 m_1 m_2 \nu}$, which differ unless $\rho_1 = \rho_2$. A message is randomized by raising $\vec{T}$ to some μ and $\vec{N}$ to an independent ν, which raises $\vec{M}$

to $\mu\nu$. The relation $e(M_j, \hat{P}) = e(T_j, N_j)$ still ties the $\mathbb{G}_1$-part to the $\mathbb{G}_2$-part, and class-hiding holds under DDH [40].

Why hashes. The group element h, which is also the first signature component, must be chosen with care. Recall that a signature under $\mathsf{sk} = (x, y_1, y_2, z_1, z_2)$ has the form $\sigma = (h, b, s) = (h, T_1^{z_1} T_2^{z_2}, h^x M_1^{y_1} M_2^{y_2})$ (Section 1). A single signer could sample a fresh h for each signature, but threshold signers, like the aggregated signers of [40], must use the same h without interacting. So h must be derived from the message itself, and it must be bound to it. Otherwise, signatures that share h can be combined. For example, if two messages $(\vec{M}, \vec{N})$ and $(\vec{M}', \vec{N}')$ with the same tag $\vec{T}$ were signed with the same h, giving (h, b, s) and (h, b', s'), then (h^2, bb', ss') would be a valid signature for the tag $\vec{T}^2$ on $(\vec{M}\vec{M}', (\vec{N}\vec{N}')^{1/2})$ (all operations entrywise), whose $\mathbb{G}_2$-part is in a new class. We therefore let h be a hash of the public part $(P^{\rho_j})_j$ of the tag secret, the $\mathbb{G}_2$-part $\vec{N}$, and the signer's public key pk. All signers compute the same h for the same tagged message, while different tagged messages, and signatures under different keys, never share h by collision resistance.

Concretely, to encode a message secret $\vec{m}$ for a signer with public key pk, one samples a tag secret $\vec{\rho}$, computes $\vec{N} = (\hat{P}^{m_1}, \ldots, \hat{P}^{m_\ell})$ and $h = H(P^{\rho_1} \| \ldots \| P^{\rho_\ell} \| \vec{N} \| \mathsf{pk})$, and outputs the tag $\vec{T} = (h^{\rho_1}, \ldots, h^{\rho_\ell})$ and $\vec{M} = (T_1^{m_1}, \ldots, T_\ell^{m_\ell})$. This is the algorithm $\mathsf{GenTagMsg}$ of Figure 3, and $\mathsf{VerifyMsg}$ checks its output. The signer needs $\vec{\rho}$ to recompute h and check the message, but signature verification uses only the group elements $\vec{T}$, $(\vec{M}, \vec{N})$, and σ, and does not recompute the hash.

Definitions 1 and 2 give the tag and message spaces. The tag space is parameterized by the $\mathbb{G}_2$-part $\vec{N}$ of a message, and a valid tagged message satisfies $(\vec{M}, \vec{N}) \in \mathcal{M}$ and $\vec{T} \in \mathcal{T}^{\vec{N}}$. Since tags are shown together with signatures, anonymity requires that tags can be randomized, which is the role of γ in Definition 1.

Definition 1 (Tag Space ($\mathcal{T}^{\vec{N}}$) [40]). *$\mathcal{T}^{\vec{N}} \subset (\mathbb{G}_1)^\ell$ is a tag space if for all $\vec{T} = (T_1, \ldots, T_\ell) \in \mathcal{T}^{\vec{N}}$, $\exists\ \gamma \in \mathbb{Z}_p^\times$, $\vec{\rho} = (\rho_1, \ldots, \rho_\ell)$ s.t. $\vec{N} = (N_1, \ldots, N_\ell) \in (\mathbb{G}_2)^\ell$, $h = H(P^{\rho_1} || \ldots || P^{\rho_\ell} || N_1 || \ldots || N_\ell || \mathsf{pk})$ and $\vec{T} = (h^{\gamma\rho_1}, \ldots, h^{\gamma\rho_\ell})$, for the signer's key pk, which we leave implicit in the notation.*

Definition 2 (Tag-based DH Message Space ($\mathcal{M}$) [40]). *$\mathcal{M} \subset (\mathbb{G}_1)^\ell \times (\mathbb{G}_2)^\ell$ is a tag-based Diffie-Hellman (DH) message space if for all $(\vec{M}, \vec{N}) = (M_1, \ldots, M_\ell, N_1, \ldots, N_\ell) \in \mathcal{M}$, $\exists \vec{T} \in \mathcal{T}^{\vec{N}}$ s.t. $\forall\ i \in [\ell]$, $e(M_i, \hat{P}) = e(T_i, N_i)$.*

We write $\mathcal{T}$, without superscript, for the union of the tag spaces $\mathcal{T}^{\vec{N}}$ over all $\vec{N}$.

We omit a review of non-threshold mercurial signature definitions here and instead present only our new definition of threshold mercurial signatures in the following section, since their description is similar. A formal description of non-threshold mercurial signatures is available in Appendix A.2. Threshold mercurial signatures imply non-threshold mercurial signatures, as can be seen by setting $n = t = 1$.

3 Threshold Mercurial Signatures

We first formally define the notion of a non-interactive threshold mercurial signature (TMS) in Section 3.1. Then in Section 3.2, we present the (non-threshold) mercurial signature construction of [40], along with our modifications that support the signing of public keys. We use this to construct TMS, and present our scheme and its security theorems in Section 3.3. Proofs omitted from the body appear in the appendix.

3.1 TMS Syntax and Security Definition

Intuition of threshold signing. A TMS distributes the signing key of a mercurial signature among n parties so that any t of them can sign, while any fewer than t learn nothing about the key. Key generation secret-shares the signing key (via a trusted dealer, or a distributed key generation protocol). To obtain a signature, a user requests a partial signature from each of t signers via $\mathsf{ParSign}$; the scheme is *non-interactive* in that the signers never communicate with each other. The user then combines the t partial signatures via $\mathsf{Reconst}$ into an ordinary mercurial signature, which verifies under the global public key exactly as a non-threshold signature would.

Intuition of randomization features. The main feature of a mercurial signature scheme is its randomizability. A signature holder can randomize the public key, message, and signature so that they are unlinkable to the originals while still verifying. In DAC, this is what allows a credential chain to be shown without revealing its intermediate issuers. Specifically, mercurial signature schemes provide a $\mathsf{ChangeRep}$ function that takes a verifying message and signature pair and outputs a randomized pair that still verifies, a $\mathsf{ConvertTag}$ function that randomizes tags, a $\mathsf{ConvertPK}$ function that randomizes a public key, and a $\mathsf{ConvertSig}$ function that updates associated signatures to verify under the new key.

Intuition of unforgeability. A definitional problem arises when message randomization is allowed: what now constitutes a forgery? If arbitrary randomizations were allowed (*i.e.*, if the image of $\mathsf{ChangeRep}(\mathsf{pk}, (\vec{M}, \vec{N}), \sigma)$ was the entire message space), then it would be impossible to define unforgeability, since a signature on any message could be updated into a signature on any other message in $\mathcal{M}$. This issue is resolved by introducing *equivalence classes*, which specify which randomizations are allowed and do not constitute a forgery. In this framework, the image of $\mathsf{ChangeRep}(\mathsf{pk}, (\vec{M}, \vec{N}), \sigma)$ is restricted to the equivalence class of $(\vec{M}, \vec{N})$, denoted $[(\vec{M}, \vec{N})]_{\mathcal{R}_\mathcal{M}}$. Unforgeability (Definition 5) then requires that the $\mathbb{G}_2$-part of the forged message not belong to the equivalence class of the $\mathbb{G}_2$-part of any message queried to the signing oracle. In the threshold setting, a message counts as queried as soon as the adversary obtains a partial signature on it from any honest party.

For example, in our construction, the equivalence class of a message is $[(\vec{M}, \vec{N})]_{\mathcal{R}_\mathcal{M}} = \{(\vec{M}', \vec{N}') : \exists \mu, \nu \in \mathbb{Z}_p^\times, (\vec{M}', \vec{N}') = (\vec{M}^\mu, \vec{N}^\nu)\}$, and the equivalence class of a tag is $[\vec{T}]_{\mathcal{R}_\mathcal{T}} = \{\vec{T}' : \exists \gamma \in \mathbb{Z}_p^\times, \vec{T}' = \vec{T}^\gamma\}$. Public keys also

have equivalence classes, which ConvertPK and public key class-hiding use. In our construction, the class of pk is $[\mathsf{pk}]_{\mathcal{R}_\mathcal{K}} = \{\mathsf{pk}^\omega : \omega \in \mathbb{Z}_p^\times\}$. The $\mathbb{G}_2$-part of a message has the same classes, $[\vec{N}]_{\mathcal{R}_\mathcal{K}} = \{\vec{N}^\nu : \nu \in \mathbb{Z}_p^\times\}$. To be signed, a key is encoded as a message with a tag (Section 3.2), and this encoding ranges over the message and tag classes.

Intuition of randomization security definitions. Given our notion of equivalence classes, we can now describe the security definitions pertaining to randomization. Message class-hiding (Definition 6) ensures that, given a message $(\vec{M}, \vec{N})$, it is difficult to distinguish a randomization $(\vec{M}', \vec{N}') \xleftarrow{\$} [(\vec{M}, \vec{N})]$ from a freshly sampled $(\vec{M}', \vec{N}') \xleftarrow{\$} \mathcal{M}$; tag class-hiding is analogous. Public key class-hiding (Definition 6) is where thresholdization matters. In a regular mercurial signature, the adversary is not a signer, or else they could use their secret key to win the game. In our thresholdized definition, the adversary corrupts up to $t-1$ signers, learning their secret keys, and can additionally query for signatures from honest signers, but still cannot distinguish a randomization of its own global public key from a fresh one. Finally, origin-hiding (Definition 7) ensures that each randomization algorithm outputs a uniformly random element of the respective equivalence class (or, for ConvertSig, the unique signature with the same h that verifies under the converted key).

We now proceed with the syntax and security definitions of threshold mercurial signatures.

Definition 3 (Threshold Mercurial Signatures). *A threshold mercurial signature scheme* TMS *is parameterized by tag, message, and key spaces $\mathcal{T}$, $\mathcal{M}$, and $\mathcal{K}$ with equivalence relations $\mathcal{R}_\mathcal{T}$, $\mathcal{R}_\mathcal{M}$, and $\mathcal{R}_\mathcal{K}$. It consists of the following* PPT *algorithms; all except* Setup *take* pp *as an implicit input.*

- $\mathsf{Setup}(1^\lambda, 1^\ell) \to \mathsf{pp}$*, the setup algorithm, outputs public parameters for messages of length ℓ.*
- $\mathsf{KeyGen}(\mathsf{pp}, n, t) \to (\vec{\mathsf{sk}}, \vec{\mathsf{pk}}, \mathsf{pk}_0)$*, the key generation algorithm, shares a global signing key sk_0 among n parties with threshold $1 \le t \le n$. It outputs the shares $\vec{\mathsf{sk}} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$, the partial public keys $\vec{\mathsf{pk}} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, and the global public key pk_0; party P_i receives $(\mathsf{sk}_i, \mathsf{pk}_i, \mathsf{pk}_0)$.*
- $\mathsf{GenTagMsg}(\vec{m}, \mathsf{pk}_0) \to (\vec{\rho}, \vec{T}, (\vec{M}, \vec{N}))$*, the tag and message generation algorithm, encodes a message secret $\vec{m} \in (\mathbb{Z}_p^\times)^\ell$ for the signer key pk_0 as a tag secret $\vec{\rho}$, a tag $\vec{T}$, and a message $(\vec{M}, \vec{N}) \in \mathcal{M}$ (Section 2).*
- $\mathsf{VerifyMsg}(\mathsf{pk}_0, \vec{\rho}, (\vec{M}, \vec{N})) \to \{0,1\}$*, the message verification algorithm, checks that $(\vec{M}, \vec{N})$ is well formed with respect to $\vec{\rho}$ and pk_0.*
- $\mathsf{ParSign}(\mathsf{sk}_i, \mathsf{pk}_0, \vec{\rho}, (\vec{M}, \vec{N})) \to \sigma_i$*, the partial signing algorithm, outputs the partial signature of party i.*
- $\mathsf{ParVerify}(\mathsf{pk}_i, \vec{T}, (\vec{M}, \vec{N}), \sigma_i) \to \{0,1\}$*, the partial verification algorithm, checks σ_i under the partial public key pk_i.*
- $\mathsf{Reconst}(\vec{\mathsf{pk}}, \vec{T}, (\vec{M}, \vec{N}), \{(i, \sigma_i)\}_{i \in \mathfrak{T}}) \to \sigma/\perp$*, the reconstruction algorithm, combines the partial signatures of a set $\mathfrak{T} \subseteq [n]$ of t parties into σ.*

- $\mathsf{Verify}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma) \to \{0,1\}$, *the verification algorithm, checks* σ *under the global public key* pk_0.
- $\mathsf{ConvertTag}(\vec{T}; \gamma) \to \vec{T}'$, *the tag conversion algorithm, outputs* $\vec{T}' \in [\vec{T}]_{\mathcal{R}_\mathcal{T}}$.
- $\mathsf{ConvertPK}(\mathsf{pk}_0; \omega) \to \mathsf{pk}_0'$, *the key conversion algorithm, yields* $\mathsf{pk}_0' \in [\mathsf{pk}_0]_{\mathcal{R}_\mathcal{K}}$.
- $\mathsf{ConvertSig}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma; \omega) \to \sigma'$, *the signature conversion algorithm, outputs a signature that verifies under* $\mathsf{ConvertPK}(\mathsf{pk}_0; \omega)$.
- $\mathsf{ChangeRep}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma; (\mu, \nu)) \to (\vec{T}', (\vec{M}', \vec{N}'), \sigma')$, *the change-of-representative algorithm, outputs* $\vec{T}' \in [\vec{T}]_{\mathcal{R}_\mathcal{T}}$, $(\vec{M}', \vec{N}') \in [(\vec{M}, \vec{N})]_{\mathcal{R}_\mathcal{M}}$, *and a signature* σ' *on them.*

KeyGen *is run by a trusted dealer, who erases* sk_0 *afterwards. If* pk_0 *is to be signed at the level above using the key* pk' *(Definition 8), the dealer first encodes it as a message with* $\mathsf{GenTagMsg}(\mathsf{sk}_0, \mathsf{pk}')$. *A distributed key generation protocol can replace the dealer. A* TMS *scheme must satisfy correctness (Definition 4), unforgeability (Definition 5), class-hiding (Definition 6), and origin-hiding (Definition 7).*

Definition 4 (Correctness). *A* TMS *scheme is correct if, for all* $\lambda, \ell, n, t \in \mathbb{N}$ *with* $1 \le t \le n$, $\mathsf{pp} \in \mathsf{Setup}(1^\lambda, 1^\ell)$, $(\vec{\mathsf{sk}}, \vec{\mathsf{pk}}, \mathsf{pk}_0) \in \mathsf{KeyGen}(\mathsf{pp}, n, t)$, $\vec{m} \in (\mathbb{Z}_p^\times)^\ell$, *and* $(\vec{\rho}, \vec{T}, (\vec{M}, \vec{N})) \in \mathsf{GenTagMsg}(\vec{m}, \mathsf{pk}_0)$, *the following hold.*

- *Partial verification. For all* $i \in [n]$ *and* $\sigma_i \in \mathsf{ParSign}(\mathsf{sk}_i, \mathsf{pk}_0, \vec{\rho}, (\vec{M}, \vec{N}))$, $\mathsf{ParVerify}(\mathsf{pk}_i, \vec{T}, (\vec{M}, \vec{N}), \sigma_i) = 1$.
- *Threshold verification. For all* $\mathfrak{T} \subseteq [n]$ *with* $|\mathfrak{T}| = t$, $\{\sigma_i \in \mathsf{ParSign}(\mathsf{sk}_i, \mathsf{pk}_0, \vec{\rho}, (\vec{M}, \vec{N}))\}_{i \in \mathfrak{T}}$, *and* $\sigma \in \mathsf{Reconst}(\vec{\mathsf{pk}}, \vec{T}, (\vec{M}, \vec{N}), \{(i, \sigma_i)\}_{i \in \mathfrak{T}})$, $\mathsf{Verify}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma) = 1$.
- *Key and signature conversion. For all* $\omega \in \mathbb{Z}_p^\times$ *and* σ *with* $\mathsf{Verify}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma) = 1$, $\mathsf{pk}_0' = \mathsf{ConvertPK}(\mathsf{pk}_0; \omega) \in [\mathsf{pk}_0]_{\mathcal{R}_\mathcal{K}}$, *and* $\sigma' \in \mathsf{ConvertSig}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma; \omega)$, $\mathsf{Verify}(\mathsf{pk}_0', \vec{T}, (\vec{M}, \vec{N}), \sigma') = 1$.
- *Change of representative. For all* $\sigma, \mu, \nu \in \mathbb{Z}_p^\times$ *and* $(\vec{T}', (\vec{M}', \vec{N}'), \sigma') \in \mathsf{ChangeRep}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma; (\mu, \nu))$, $\mathsf{Verify}(\mathsf{pk}_0, \vec{T}', (\vec{M}', \vec{N}'), \sigma') = 1$, $\vec{T}' \in [\vec{T}]_{\mathcal{R}_\mathcal{T}}$, *and* $(\vec{M}', \vec{N}') \in [(\vec{M}, \vec{N})]_{\mathcal{R}_\mathcal{M}}$.

Definition 5 (Threshold Unforgeability). *A* TMS *scheme is existentially unforgeable under chosen tagged message attack (*$\mathsf{EUF\text{-}CtMA}$*) if, for all* $\lambda, n, t \in \mathbb{N}$ *with* $1 \le t \le n$ *and all* PPT *adversaries* $\mathcal{A}$, $\Pr\left[\mathsf{EUF\text{-}CtMA}_\mathcal{A}^{\mathsf{TMS}}(1^\lambda) = 1\right] \le \mathsf{negl}(\lambda)$, *where the game is defined in Figure 1.*

A forgery may be under any key in $[\mathsf{pk}_0]_{\mathcal{R}_\mathcal{K}}$, as, when applied to delegatable anonymous credentials, a credential is shown under a randomized key. We only count forgeries if $\vec{N}^*$ is outside the class $[\vec{N}]_{\mathcal{R}_\mathcal{K}}$ of every queried $\vec{N}$. We do not consider forgeries which differ only on tag equivalence classes, $[\vec{T}]$. This is because in DAC, we want showings of intermediate issuers to contain only the public parts of tags and to enforce that tag equivalence classes are unforgeable

requires showing of the tag secret due to the similarity of our construction to [40]. Enforcing unforgeability on the $\vec{N}^*$ parts of messages is enough to construct meaningful applications as this part of the message can hold user data such as their identity and attributes.

The adversary corrupts signers statically, before key generation. In the terminology of [9], this is TS-UF-0 with static corruptions. Stronger notions (TS-UF-1, adaptive corruptions), as achieved for threshold structure-preserving signatures in [42], are left for future work. In the cross-scheme correct version (the boxed elements of Figure 1), the adversary also sees the challenge key encoded as a message for a higher-level key pk' of its choosing. There, sk_0 is the signing key shared by $\vec{\mathsf{sk}}$, and $\mathsf{GenTagMsg}_{\mathsf{pp}'}$ runs $\mathsf{GenTagMsg}$ with $\mathsf{pp}' = \mathsf{ExtendSetup}(\mathsf{pp})$ (Section 3.1).

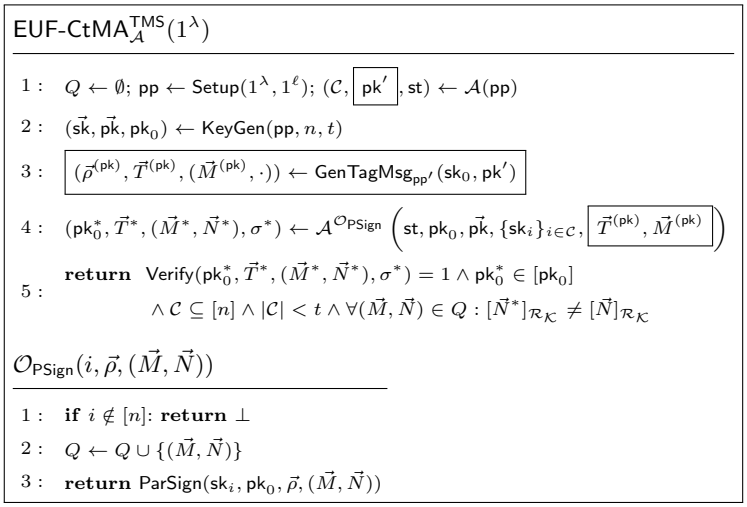

Fig. 1: Unforgeability game $\mathsf{EUF\text{-}CtMA}^{\mathsf{TMS}}_{\mathcal{A}}(1^\lambda)$ for a TMS scheme. Boxed elements belong only to the cross-scheme correct version.

Definition 6 (Class-Hiding). *A* TMS *scheme is message (resp. tag, threshold public key) class-hiding if, for all $\lambda, n, t \in \mathbb{N}$ with $1 \leq t \leq n$ and all* PPT *adversaries $\mathcal{A}$,* $\Pr\left[\mathsf{G}^{\mathsf{TMS}}_{\mathcal{A}}(1^\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$ *for* $\mathsf{G} = \mathsf{MSG\text{-}CLH}$ *(resp.* TAG-CLH, PK-CLH*), defined in Figure 2.*

In the $b = 1$ branch of PK-CLH, the challenger binds each partial signature to the converted key $\mathsf{pk}^1_{2,0}$ and then converts the reconstructed signature with ω. Since ConvertSig leaves h unchanged, this yields exactly the signature that the converted key would produce, and no party ever holds shares of the converted key. In the cross-scheme correct version, the adversary also sees both challenge keys encoded as messages for a higher-level key pk' of its choosing, as a showing

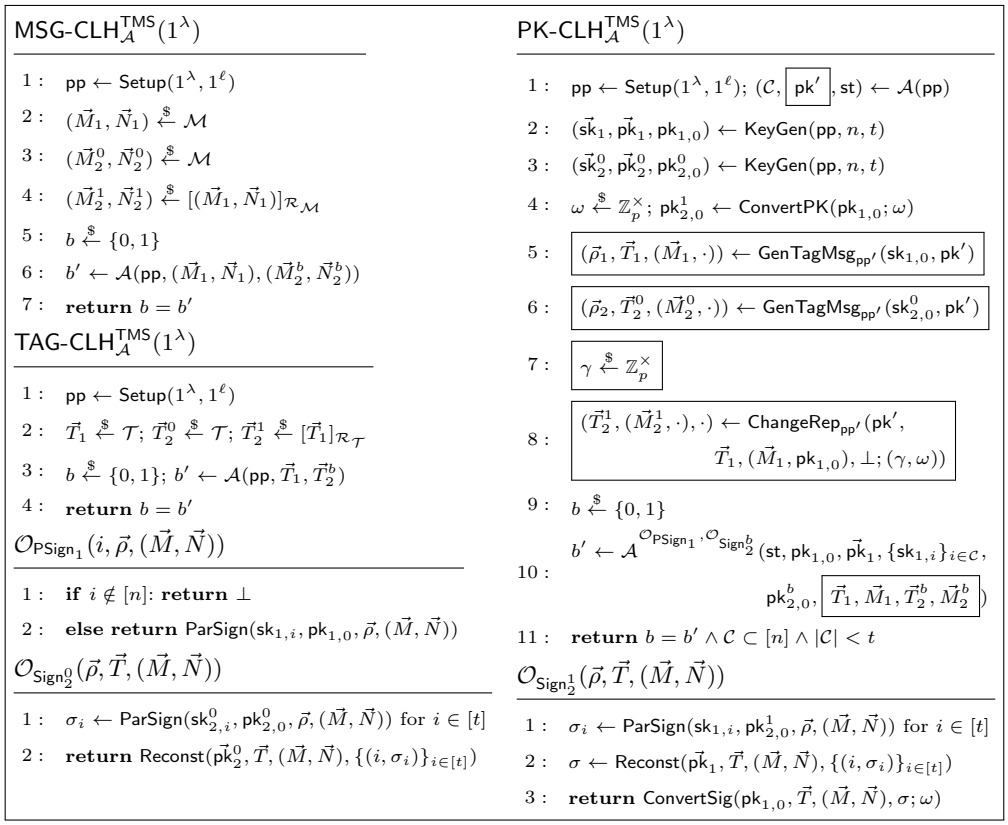

Fig. 2: Message, tag, and public key class-hiding games for a TMS scheme. Boxed elements belong only to the cross-scheme correct version.

would reveal them. For $b = 1$, the encoding of $\mathsf{pk}^1_{2,0}$ is the one that ChangeRep derives from the encoding of $\mathsf{pk}_{1,0}$. The tag and message outputs of ChangeRep do not depend on the signature, which is therefore $\perp$ in Figure 2.

Definition 7 (Origin-Hiding). *A* TMS *scheme is origin-hiding if, for all* pp, $(\vec{\mathsf{sk}}, \vec{\mathsf{pk}}, \mathsf{pk}_0) \in \mathsf{KeyGen}(\mathsf{pp}, n, t)$, $(\vec{\rho}, \vec{T}, (\vec{M}, \vec{N})) \in \mathsf{GenTagMsg}(\vec{m}, \mathsf{pk}_0)$, *and* σ *with* $\mathsf{Verify}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma) = 1$*:*

- *for* $\omega \xleftarrow{\$} \mathbb{Z}_p^\times$, $\mathsf{ConvertPK}(\mathsf{pk}_0; \omega)$ *is uniform in* $[\mathsf{pk}_0]_{\mathcal{R}_\mathcal{K}}$, *and* $\mathsf{ConvertSig}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma; \omega)$ *is the unique signature that has the same first component* h *as* σ *and verifies under* $\mathsf{ConvertPK}(\mathsf{pk}_0; \omega)$*;*
- *for* $(\mu, \nu) \xleftarrow{\$} (\mathbb{Z}_p^\times)^2$ *and* $\gamma \xleftarrow{\$} \mathbb{Z}_p^\times$, *the outputs* $\vec{T}'$ *and* $(\vec{M}', \vec{N}')$ *of* $\mathsf{ChangeRep}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma; (\mu, \nu))$ *are uniform in* $[\vec{T}]_{\mathcal{R}_\mathcal{T}}$ *and* $[(\vec{M}, \vec{N})]_{\mathcal{R}_\mathcal{M}}$, *and* $\mathsf{ConvertTag}(\vec{T}; \gamma)$ *is uniform in* $[\vec{T}]_{\mathcal{R}_\mathcal{T}}$.

Setting $n = t = 1$ yields the corresponding definitions for (non-threshold) mercurial signatures (Appendix A.2).

Cross-scheme correctness: Mercurial signatures such as [40] do not support the signing of public keys as messages. This was an informal property of [20], later formalized as *cross-scheme correctness* [34]: the scheme can be extended to a new scheme whose message space contains the original key space, and the extended scheme can itself be extended, so a key at any level of a delegation chain can be signed by a key at the level above. We define a leveled variant for TMS, with an algorithm $\mathsf{ExtendSetup}(\mathsf{pp}) \to \mathsf{pp}'$ that outputs the parameters of the level above. We write $\mathsf{Alg}_{\mathsf{pp}'}$ for an algorithm run with parameters pp'. A key pk_0 generated under pp is signed under pp' like any other message: the holder of its signing key sk_0 runs $\mathsf{GenTagMsg}_{\mathsf{pp}'}$ on the message secret sk_0 and key pk' that will do the signing, which yields a message whose $\mathbb{G}_2$-part is pk_0.

Definition 8 (Leveled cross-scheme correctness). *A* TMS *scheme is leveled cross-scheme correct if there is a* PPT *algorithm* ExtendSetup *such that, for all* $\lambda, \ell, n, t \in \mathbb{N}$ *with* $1 \le t \le n$, $\mathsf{pp} \in \mathsf{Setup}(1^\lambda, 1^\ell)$, *and* $\mathsf{pp}' \in \mathsf{ExtendSetup}(\mathsf{pp})$, *it holds that* $\mathsf{pp}' \in \mathsf{Setup}(1^\lambda, 1^{\ell'})$ *for some* ℓ', *and, for all* $(\vec{\mathsf{sk}}, \vec{\mathsf{pk}}, \mathsf{pk}_0) \in \mathsf{KeyGen}(\mathsf{pp}, n, t)$ *with shared signing key* sk_0, *all* $(\vec{\mathsf{sk}}', \vec{\mathsf{pk}}', \mathsf{pk}'_0) \in \mathsf{KeyGen}(\mathsf{pp}', n, t)$, *and all* $(\vec{\rho}, \vec{T}, (\vec{M}, \vec{N})) \in \mathsf{GenTagMsg}_{\mathsf{pp}'}(\mathsf{sk}_0, \mathsf{pk}'_0)$, *it holds that* $\vec{N} = \mathsf{pk}_0$. *By correctness under* pp', *any* t *holders of shares of* sk'_0 *can then sign* pk_0, *completed by* $(\vec{T}, \vec{M})$ *to a message.*

3.2 Enhanced MS Construction (adapted from [40])

Before presenting our threshold construction, we recall the (non-threshold) mercurial signature of [40] on which it is based. We call it $\mathbf{\Pi}_{\mathsf{MBGLS}}$, after the authors of [40], and present it in Figure 3 for messages of length $\ell = 2$ (Figure 6 with $n = t = 1$ gives general ℓ). We make four changes to [40].

- We drop the aggregation algorithms (AggrSign and VerifyAggr), which we do not use. We drop VerifyTag, which our modified unforgeability definition (Definition 5) no longer uses.
- In [40], signing takes auxiliary data aux that lists the public keys of all signers whose signatures will be aggregated. The signer checks that its own key appears in aux, and h is a hash of $(P^{\rho_j})_j$, $\vec{N}$, and aux. Without aggregation, aux contains only the signer's key, so we pass the signer's public key pk directly and hash it in place of aux; with a single signer, this is equivalent to [40]. Binding h to the signer's key ensures that signatures under different keys, including randomizations of the same key, never share h.
- We add ExtendSetup so that public keys can themselves be signed (cross-scheme correctness). We call the scheme without the elements in solid boxes (in Figures 1 and 2) the *plain* version and the scheme with them the *cross-scheme correct* version. Equivalence classes are as in Section 3.1 (formally in Appendix B).

- $\mathsf{VerifyMsg}$ rejects tag secrets with a zero entry, and Verify rejects tags and messages in which some T_j or N_j is the identity, since otherwise the key encoding below would yield forgeries.

Since $\mathsf{pk} = \hat{P}^{\mathsf{sk}}$ (entrywise), a public key is the $\mathbb{G}_2$-part of the message with message secret sk. To have pk signed under a key pk' of the level above, its owner runs $\mathsf{GenTagMsg}_{\mathsf{pp}'}(\mathsf{sk}, \mathsf{pk}')$ with $\mathsf{pp}' = \mathsf{ExtendSetup}(\mathsf{pp})$ and obtains $(\vec{\rho}^{(\mathsf{pk})}, \vec{T}^{(\mathsf{pk})}, (\vec{M}^{(\mathsf{pk})}, \mathsf{pk}))$, where $T_j^{(\mathsf{pk})} = h^{\rho_j}$, $M_j^{(\mathsf{pk})} = (T_j^{(\mathsf{pk})})^{\mathsf{sk}_j}$, and $h = \mathsf{H}(P^{\rho_1} \| \ldots \| P^{\rho_5} \| \mathsf{pk} \| \mathsf{pk}')$. The difference between the plain and cross-scheme correct versions is not in the algorithms, but in the security games. In the cross-scheme version $(\vec{T}^{(\mathsf{pk})}, \vec{M}^{(\mathsf{pk})})$ becomes public, and our security games for this version (Figures 1 and 2) give it to the adversary. A signature on pk under pk' is updated to one on $\mathsf{ConvertPK}(\mathsf{pk}; \omega) = \mathsf{pk}^{\omega}$ by running $\mathsf{ChangeRep}$ with $(\mu, \nu) = (\gamma, \omega)$ for any γ, which maps the encoding to $((\vec{T}^{(\mathsf{pk})})^{\gamma}, (\vec{M}^{(\mathsf{pk})})^{\gamma\omega}, \mathsf{pk}^{\omega})$.

General ℓ and $\mathsf{ExtendSetup}$. For general ℓ, keys have $2\ell + 1$ components, $\mathsf{sk}_0 = (x, y_1, \ldots, y_\ell, z_1, \ldots, z_\ell)$ (Figure 6). A public key under pp is thus the $\mathbb{G}_2$-part of a message of length $2\ell + 1$, and $\mathsf{ExtendSetup}$ (Figure 3) replaces ℓ by $2\ell + 1$.

Theorem 1. $\mathbf{\Pi}_{\mathsf{MBGLS}}$ *is correct (Definition 4 with $n = t = 1$) and the cross-scheme version is leveled cross-scheme correct (Definition 8).* $\mathbf{\Pi}_{\mathsf{MBGLS}}$ *is message and tag class-hiding (Definition 6) assuming DDH holds in $\mathbb{G}_1$ and $\mathbb{G}_2$, and perfectly origin-hiding (Definition 7).*

Definitions 11 and 12 (Appendix A.2) write out the games of Figures 1 and 2 for $n = t = 1$. Theorem 1 follows from [40] for both versions, since the cross-scheme correct version has the same algorithms, plus $\mathsf{ExtendSetup}$. Leveled cross-scheme correctness holds because $\mathsf{GenTagMsg}_{\mathsf{pp}'}(\mathsf{sk}, \mathsf{pk}')$ outputs a message whose $\mathbb{G}_2$-part is $\mathsf{pk} = \hat{P}^{\mathsf{sk}}$.

Theorem 2. $\mathbf{\Pi}_{\mathsf{MBGLS}}$ *is* $\mathsf{EUF\text{-}CtMA}$ *(Definition 11) in the GGM and ROM.*

Proof idea. Since our unforgeability game differs from that of [40], we prove Theorem 2 directly, for both versions (Appendix C). In the GGM, the adversary obtains group elements only through oracles, so the discrete log of each element it holds is a polynomial in the secret values, determined by its queries. These secret values are the signing key, the hash outputs, and, in the cross-scheme correct version, the tag secrets of the key encoding. A forgery must satisfy the verification equations as polynomial identities. First, consider a signing query on a message with tag secret $\vec{\rho}$. The signer recomputes the hash h and checks that $M_j = h^{\rho_j n_j}$, where n_j is the discrete log of N_j. Since h is a fresh random element, the adversary can involve h in a group element only by raising h to a constant that it knows, so each n_j must be such a constant. Thus the adversary obtains signatures only on messages whose discrete logs it knows, and never on messages derived from the public key, which fixes the form of every signature it receives. Next, the second verification equation, which involves only the tag, shows that the forged tag is a product of powers of queried tags, with the same

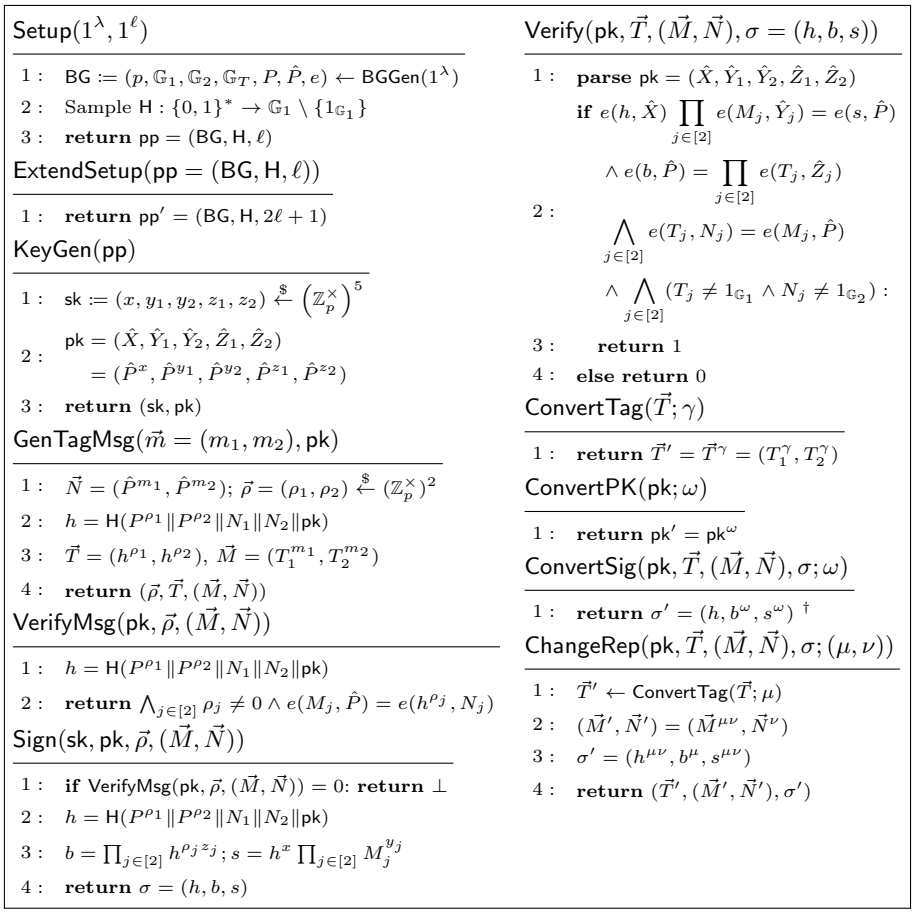

Fig. 3: Mercurial signature scheme $\mathbf{\Pi}_{\mathsf{MBGLS}}$ (modified from [40]) for $\ell = 2$. All algorithms take pp as an implicit input.

† ConvertSig leaves h unchanged; in our TDAC it is always followed by ChangeRep, which rerandomizes h and the tag.

exponent in each entry, and possibly of elements of the key encoding. The first and third equations rule out the key encoding, since $\vec{N}^*$ has no identity entries. Finally, the first equation forces $\vec{N}^*$ to be a power of some queried $\vec{N}^{(k)}$, so the forgery fails the class condition.

Theorem 3. $\mathbf{\Pi}_{\mathsf{MBGLS}}$ *is public key class-hiding (Definition 12) in the GGM and ROM.*

Proof idea. For the plain version, this is shown in [40]. For the cross-scheme correct version, we reduce to the plain version (Appendix D.2). The adversary additionally sees the key encodings $(\vec{T}, \vec{M})$, which the reduction cannot compute

without the secret keys, so we must show that they do not help the adversary tell $b = 0$ from $b = 1$. If a generic adversary distinguishes the two cases, an extractor [35] finds a polynomial, formed from its queries, that is zero in one case and nonzero in the other. We show that this polynomial does not depend on the key encodings, so the reduction can compute it in its own game and use it to distinguish.

3.3 Construction of Threshold Mercurial Signatures

Figure 4 presents our threshold mercurial signature scheme $\mathbf{\Pi}_{\mathsf{TMS}}$ for $\ell = 2$; Figure 6 in Appendix B.1 handles general $\ell \in \mathsf{poly}(\lambda)$. $\mathbf{\Pi}_{\mathsf{TMS}}$ combines Shamir secret sharing (Section 2) with $\mathbf{\Pi}_{\mathsf{MBGLS}}$ (Figure 3). The dealer secret-shares a $\mathbf{\Pi}_{\mathsf{MBGLS}}$ signing key, each signer runs $\mathbf{\Pi}_{\mathsf{MBGLS}}.\mathsf{Sign}$ with its key-share and the global public key, and $\mathsf{Reconst}$ interpolates the partial signatures in the exponent. All partial signatures on a message have the same h, since h depends only on the message, the tag secret, and the global public key. The algorithms not shown in Figure 4 (Setup, $\mathsf{ExtendSetup}$, $\mathsf{GenTagMsg}$, $\mathsf{VerifyMsg}$, Verify, $\mathsf{ConvertTag}$, $\mathsf{ConvertPK}$, $\mathsf{ConvertSig}$, and $\mathsf{ChangeRep}$) are the same as in $\mathbf{\Pi}_{\mathsf{MBGLS}}$.

$\mathsf{KeyGen}(\mathsf{pp}, n, t)$

1 : $(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathbf{\Pi}_{\mathsf{MBGLS}}.\mathsf{KeyGen}(\mathsf{pp})$
2 : $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n) \leftarrow \mathsf{SSS.Share}(\mathsf{sk}_0, n, t)$
3 : $\mathsf{pk}_i = \hat{P}^{\mathsf{sk}_i}$ (entrywise) for $i \in [n]$
4 : **return** $(\vec{\mathsf{sk}}, \vec{\mathsf{pk}}, \mathsf{pk}_0)$
5 : // P_i gets $(\mathsf{sk}_i, \mathsf{pk}_i, \mathsf{pk}_0)$; erase sk_0

$\mathsf{ParSign}(\mathsf{sk}_i, \mathsf{pk}_0, \vec{\rho}, (\vec{M}, \vec{N}))$

1 : **return** $\mathbf{\Pi}_{\mathsf{MBGLS}}.\mathsf{Sign}(\mathsf{sk}_i, \mathsf{pk}_0, \vec{\rho}, (\vec{M}, \vec{N}))$

$\mathsf{ParVerify}(\mathsf{pk}_i, \vec{T}, (\vec{M}, \vec{N}), \sigma_i)$

1 : **return** $\mathbf{\Pi}_{\mathsf{MBGLS}}.\mathsf{Verify}(\mathsf{pk}_i, \vec{T}, (\vec{M}, \vec{N}), \sigma_i)$

$\mathsf{Reconst}(\vec{\mathsf{pk}}, \vec{T}, (\vec{M}, \vec{N}), \{(i, \sigma_i)\}_{i \in \mathfrak{T}})$

1 : **if** $\mathfrak{T} \not\subseteq [n] \vee |\mathfrak{T}| \neq t$: **return** $\perp$
2 : **parse** $\sigma_i = (h_i, b_i, s_i)$ for $i \in \mathfrak{T}$
3 : **if** $h_i \neq h_j$ for some $i, j \in \mathfrak{T}$: **return** $\perp$
4 : **if** $\mathsf{ParVerify}(\mathsf{pk}_i, \vec{T}, (\vec{M}, \vec{N}), \sigma_i) = 0$
5 : for some $i \in \mathfrak{T}$: **return** $\perp$
6 : **return** $\sigma = \left(h_i, \prod_{i \in \mathfrak{T}} b_i^{\lambda_{i,\mathfrak{T}}}, \prod_{i \in \mathfrak{T}} s_i^{\lambda_{i,\mathfrak{T}}}\right)$

Fig. 4: Threshold mercurial signature scheme $\mathbf{\Pi}_{\mathsf{TMS}}$ for $\ell = 2$.

Theorem 4 (Correctness and randomization). $\mathbf{\Pi}_{\mathsf{TMS}}$ *is correct, and its cross-scheme correct version (with* $\mathsf{ExtendSetup}$ *from Figure 3) is leveled cross-scheme correct. Assuming DDH holds in* $\mathbb{G}_1$ *and* $\mathbb{G}_2$, $\mathbf{\Pi}_{\mathsf{TMS}}$ *is message and tag class-hiding; it is perfectly origin-hiding.*

Proof. Messages, tags, and their randomization are those of $\mathbf{\Pi}_{\mathsf{MBGLS}}$, so class-hiding and origin-hiding follow from Theorem 1. A partial signature is a $\mathbf{\Pi}_{\mathsf{MBGLS}}$ signature under $(\mathsf{sk}_i, \mathsf{pk}_i)$, except that h is computed with pk_0, and Verify does not recompute h; so partial verification, key and signature conversion, and change of representative follow by inspection, as for $\mathbf{\Pi}_{\mathsf{MBGLS}}$ [40], and the identity checks in Verify pass as for $\mathbf{\Pi}_{\mathsf{MBGLS}}$ (Section 3.2). For threshold verification, let $\sigma_i = (h_i, b_i, s_i) \in \mathsf{ParSign}(\mathsf{sk}_i, \mathsf{pk}_0, \vec{\rho}, (\vec{M}, \vec{N}))$ for $i \in \mathfrak{T}$. All h_i equal

$h = \mathsf{H}(P^{\rho_1} \| P^{\rho_2} \| N_1 \| N_2 \| \mathsf{pk}_0)$, and each σ_i passes $\mathsf{ParVerify}$, so $\mathsf{Reconst}$ outputs $\sigma = (h, b, s)$. Since $\mathsf{sk}_0 = \sum_{i \in \mathfrak{T}} \lambda_{i,\mathfrak{T}} \mathsf{sk}_i$ entrywise,

$$b = \prod_{i \in \mathfrak{T}} b_i^{\lambda_{i,\mathfrak{T}}} = \prod_{j \in [2]} h^{\rho_j \sum_{i \in \mathfrak{T}} \lambda_{i,\mathfrak{T}} z_{i,j}} = \prod_{j \in [2]} h^{\rho_j z_j},$$

$$s = \prod_{i \in \mathfrak{T}} s_i^{\lambda_{i,\mathfrak{T}}} = h^{\sum_{i \in \mathfrak{T}} \lambda_{i,\mathfrak{T}} x_i} \prod_{j \in [2]} M_j^{\sum_{i \in \mathfrak{T}} \lambda_{i,\mathfrak{T}} y_{i,j}} = h^x \prod_{j \in [2]} M_j^{y_j}.$$

This is the signature $\mathbf{\Pi}_{\mathsf{MBGLS}}.\mathsf{Sign}(\mathsf{sk}_0, \mathsf{pk}_0, \vec{\rho}, (\vec{M}, \vec{N}))$, so $\mathsf{Verify}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma) = 1$. For leveled cross-scheme correctness, $\mathsf{pp}' = \mathsf{ExtendSetup}(\mathsf{pp})$ equals pp except that the message length is $\ell' = 2\ell + 1$, so $\mathsf{pp}' \in \mathsf{Setup}(1^\lambda, 1^{\ell'})$. The shared signing key sk_0 has ℓ' components and $\mathsf{pk}_0 = \hat{P}^{\mathsf{sk}_0}$, so $\mathsf{GenTagMsg}_{\mathsf{pp}'}(\mathsf{sk}_0, \mathsf{pk}_0')$ outputs a message with $N_j = \hat{P}^{\mathsf{sk}_{0,j}}$ for $j \in [\ell']$, *i.e.*, $\vec{N} = \mathsf{pk}_0$. □

Theorem 5 (Unforgeability). *If* $\mathbf{\Pi}_{\mathsf{MBGLS}}$ *is* EUF-CtMA *(Definition 11), then so is* $\mathbf{\Pi}_{\mathsf{TMS}}$. *This holds for each version of* $\mathbf{\Pi}_{\mathsf{TMS}}$ *with respect to the corresponding version of* $\mathbf{\Pi}_{\mathsf{MBGLS}}$.

Proof. Let $\mathcal{A}$ be a PPT adversary against $\mathsf{EUF\text{-}CtMA}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{TMS}}}$ (Figure 1). We construct a reduction $\mathcal{B}$ that runs $\mathcal{A}$ and plays $\mathsf{EUF\text{-}CtMA}_{\mathcal{B}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ (Definition 11) against the plain (resp. cross-scheme correct) version of $\mathbf{\Pi}_{\mathsf{MBGLS}}$ when $\mathcal{A}$ plays against the plain (resp. cross-scheme correct) version of $\mathbf{\Pi}_{\mathsf{TMS}}$. Boxed steps below occur only in the cross-scheme correct case.

Setup. $\mathcal{B}$ receives pp from its challenger and forwards it to $\mathcal{A}$, which returns $(\mathcal{C}, \boxed{\mathsf{pk}'})$. $\mathcal{B}$ sends $\boxed{\mathsf{pk}'}$ to its challenger. Assume without loss of generality that $|\mathcal{C}| = t - 1$.

Key Generation. $\mathcal{B}$ receives pk_0 and $\boxed{\vec{T}^{(\mathsf{pk})}, \vec{M}^{(\mathsf{pk})}}$ from its challenger. For each $i \in \mathcal{C}$, it samples the partial secret keys $\mathsf{sk}_i \xleftarrow{\$} (\mathbb{Z}_p)^5$ and sets $\mathsf{pk}_i = \hat{P}^{\mathsf{sk}_i}$ (entrywise). Then for each honest $k \in \mathcal{H} = [n] \setminus \mathcal{C}$, it generates partial public keys as

$$\mathsf{pk}_k = \mathsf{pk}_0^{\lambda_{0,\tilde{\mathcal{T}}}(k)} \prod_{i \in \mathcal{C}} \mathsf{pk}_i^{\lambda_{i,\tilde{\mathcal{T}}}(k)} \quad \text{(entrywise)},$$

with $\tilde{\mathcal{T}} = \mathcal{C} \cup \{0\}$ and Lagrange coefficients computed as in Section 2. It then sends pk_0, $\vec{\mathsf{pk}} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, $\{\mathsf{sk}_i\}_{i \in \mathcal{C}}$, and $\boxed{\vec{T}^{(\mathsf{pk})}, \vec{M}^{(\mathsf{pk})}}$ to $\mathcal{A}$.

Signing. On a query $\mathcal{O}_{\mathsf{PSign}}(i, \vec{\rho}, (\vec{M}, \vec{N}))$ with $i \in [n]$, $\mathcal{B}$ queries its own signing oracle on $(\vec{\rho}, (\vec{M}, \vec{N}))$, which adds $(\vec{M}, \vec{N})$ to its set Q. If the answer is $\perp$, it returns $\perp$. Otherwise it obtains $\sigma_0 = (h, b_0, s_0)$, computes $\sigma_j = (h, b_j, s_j) \leftarrow \mathsf{ParSign}(\mathsf{sk}_j, \mathsf{pk}_0, \vec{\rho}, (\vec{M}, \vec{N}))$ for $j \in \mathcal{C}$, and returns

$$\sigma_i = \Big(h,\ b_0^{\lambda_{0,\tilde{\mathcal{T}}}(i)} \prod_{j \in \mathcal{C}} b_j^{\lambda_{j,\tilde{\mathcal{T}}}(i)},\ s_0^{\lambda_{0,\tilde{\mathcal{T}}}(i)} \prod_{j \in \mathcal{C}} s_j^{\lambda_{j,\tilde{\mathcal{T}}}(i)}\Big).$$

Output. When $\mathcal{A}$ outputs $(\mathsf{pk}_0^*, \vec{T}^*, (\vec{M}^*, \vec{N}^*), \sigma^*)$, $\mathcal{B}$ outputs the same tuple.

Analysis. $\mathcal{B}$ simulates $\mathsf{EUF\text{-}CtMA}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{TMS}}}$ perfectly:

- In the real game, the shares of sk_0 lie on a uniformly random polynomial of degree $t-1$ with constant term sk_0. Its values at the $t-1$ points in $\mathcal{C}$ are uniform and independent of sk_0, and together with sk_0 they determine it, with $\mathsf{sk}_k = \lambda_{0,\tilde{\mathcal{T}}}(k)\mathsf{sk}_0 + \sum_{i\in\mathcal{C}} \lambda_{i,\tilde{\mathcal{T}}}(k)\mathsf{sk}_i$. Hence the corrupted shares and all partial public keys are distributed as in the real game.
- The components b and s of $\mathbf{\Pi}_{\mathsf{MBGLS}}.\mathsf{Sign}$ are linear in the signing key for fixed h, $\vec{\rho}$, and $(\vec{M}, \vec{N})$, so the formula above yields exactly $\mathsf{ParSign}(\mathsf{sk}_i, \mathsf{pk}_0, \vec{\rho}, (\vec{M}, \vec{N}))$. Both return $\perp$ exactly when $\mathsf{VerifyMsg}(\mathsf{pk}_0, \vec{\rho}, (\vec{M}, \vec{N})) = 0$.
- $\mathcal{B}$ queries its oracle on exactly the messages that $\mathcal{A}$ queries, so both games have the same set of queries Q. If $\mathcal{A}$ wins, then σ^* verifies under $\mathsf{pk}_0^* \in [\mathsf{pk}_0]$, and $[\vec{N}^*]_{\mathcal{R}_\mathcal{K}}$ differs from the class of the $\mathbb{G}_2$-part of every message in Q. These are exactly the winning conditions of $\mathsf{EUF\text{-}CtMA}_{\mathcal{B}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$, whose challenge key is pk_0.

Hence $\mathcal{B}$ wins $\mathsf{EUF\text{-}CtMA}_{\mathcal{B}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ whenever $\mathcal{A}$ wins $\mathsf{EUF\text{-}CtMA}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{TMS}}}$, which proves the theorem. □

Theorem 6 (Public Key Class-Hiding). *If $\mathbf{\Pi}_{\mathsf{MBGLS}}$ is public key class-hiding (Definition 12), then $\mathbf{\Pi}_{\mathsf{TMS}}$ is threshold public key class-hiding, again for each version.*

The proof of Theorem 6 is a similar reduction and appears in Appendix D.1.

Corollary 1. *Both versions of $\mathbf{\Pi}_{\mathsf{TMS}}$ are* EUF-CtMA *and threshold public key class-hiding in the GGM and ROM.*

4 Applications to Threshold DAC

In this section, we informally describe threshold delegatable anonymous credentials (TDAC) and our construction from TMS. Due to space constraints, the formal definitions, the construction, and the security proofs are given in Appendix E.

Syntax. TDAC is an anonymous credential scheme in which a root authority delegates issuing power to intermediate issuers, who remain hidden when a credential is shown, and in which each issuer's key is shared among n parties so that any t of them can issue. It consists of the following algorithms and protocols:

- $\mathsf{Setup}(1^\lambda, 1^L, 1^n, 1^t) \to \mathsf{pp}$, the setup algorithm, generates the public parameters for credential chains of L keys, from the root down to the user, where each issuer key is shared among n parties with threshold t.

- $\mathsf{IssKeyGen}(\mathsf{pp}, L', \mathsf{pk}') \to (\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]})$, the issuer key generation algorithm, generates the public key pk of a new issuer at level L' and one secret share sk_i per party; pk' is the key of the issuer that will certify pk ($\perp$ for the root).
- $\mathsf{UserKeyGen}(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{sk})$, the user key generation algorithm, generates a user's key pair; users are not thresholdized.
- $\langle \mathsf{Issue}(\mathsf{sk}_{\mathsf{I},i}, \mathsf{cred}_\mathsf{I}, L') \leftrightarrow \mathsf{Receive}(\mathsf{sk}_{\mathsf{R},j}, L')\rangle \to \mathsf{pcred}_{\mathsf{R},i}$, the issuance protocol, gives party j of a new issuer R a partial credential from party i of issuer I.
- $\mathsf{CredComb}(\{\mathsf{pcred}_{\mathsf{R},i}\}_{i\in\mathfrak{T}}, \mathsf{pk}_\mathsf{R}) \to \mathsf{cred}_\mathsf{R}$, the credential combination algorithm, combines partial credentials from t parties of I into R's credential.
- $\langle \mathsf{UIssue}(\mathsf{sk}_{\mathsf{I},i}, \mathsf{cred}_\mathsf{I}) \leftrightarrow \mathsf{UReceive}(\mathsf{sk})\rangle \to \mathsf{pcred}_i$, the user issuance protocol, gives a user a partial credential from party i of an issuer I at level $L-1$.
- $\mathsf{UCredComb}(\{\mathsf{pcred}_i\}_{i\in\mathfrak{T}}, \mathsf{pk}) \to \mathsf{cred}$, the user credential combination algorithm, combines t of them into the user's credential.
- $\langle \mathsf{Prove}(\mathsf{sk}, \mathsf{cred}) \leftrightarrow \mathsf{Verify}(\mathsf{pk}_1)\rangle \to \{0,1\}$, the showing protocol, lets a user prove that it holds a credential whose chain starts at the root key pk_1.

A TDAC scheme must satisfy the properties of *correctness*, *anonymity*, and *unforgeability*, which we describe in Definitions 14 to 16 in Appendix E.

Usage. Consider delegation depth $L = 3$: a root authority delegates to intermediate issuers, who issue credentials to users. A credential is then a chain of public keys $\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{pk}_3$, each signed by the key before it; we index keys and their shares by level.

- *Setup.* After $\mathsf{Setup}(1^\lambda, 1^3, 1^n, 1^t) \to \mathsf{pp}$, the root runs $\mathsf{IssKeyGen}(\mathsf{pp}, 1, \perp) \to (\mathsf{pk}_1, \{\mathsf{sk}_{1,i}\}_{i\in[n]})$, via a trusted dealer or, since the root key is never signed, a DKG protocol. Each of the n parties holds one share, pk_1 is published, and the root's credential is $\mathsf{cred}_1 = (\mathsf{pk}_1, \perp)$.
- *Delegation.* An intermediate issuer runs $\mathsf{IssKeyGen}(\mathsf{pp}, 2, \mathsf{pk}_1) \to (\mathsf{pk}_2, \{\mathsf{sk}_{2,i}\}_{i\in[n]})$. One of its parties, say party j, runs $\langle \mathsf{Issue}(\mathsf{sk}_{1,i}, \mathsf{cred}_1, 2) \leftrightarrow \mathsf{Receive}(\mathsf{sk}_{2,j}, 2)\rangle \to \mathsf{pcred}_{2,i}$ with t root parties $i \in \mathfrak{T}$ and combines the results into $\mathsf{cred}_2 = \mathsf{CredComb}(\{\mathsf{pcred}_{2,i}\}_{i\in\mathfrak{T}}, \mathsf{pk}_2) = \mathsf{cred}_1 \| (\mathsf{pk}_2, \vec{T}_2, \vec{M}_2, \sigma_2)$, where $(\vec{T}_2, \vec{M}_2)$ completes pk_2 to a message and σ_2 is the root's signature on it. It then shares cred_2 with the other parties.
- *Issuance to users.* A user runs $\mathsf{UserKeyGen}(\mathsf{pp}) \to (\mathsf{pk}_3, \mathsf{sk}_3)$, runs $\langle \mathsf{UIssue}(\mathsf{sk}_{2,i}, \mathsf{cred}_2) \leftrightarrow \mathsf{UReceive}(\mathsf{sk}_3)\rangle$ with t parties of the intermediate issuer, and combines the results with $\mathsf{UCredComb}$ into $\mathsf{cred}_3 = \mathsf{cred}_2 \| (\mathsf{pk}_3, \vec{T}_3, \vec{M}_3, \sigma_3)$.
- *Showing.* The user runs $\langle \mathsf{Prove}(\mathsf{sk}_3, \mathsf{cred}_3) \leftrightarrow \mathsf{Verify}(\mathsf{pk}_1)\rangle \to \{0,1\}$. The verifier learns that the chain starts at pk_1, but not which intermediate issuer it contains.

Unforgeability. Unforgeability guarantees that no PPT adversary can show a valid credential without having been issued one. In the game, the challenger generates the root key; the adversary may have it create intermediate issuers, learning their public keys and the shares of up to $t-1$ parties of each, and users, and may see these users' showings. It wins if it then shows a credential that

verifies under pk_1; since it has no oracle access during this showing, it cannot relay an honest user's showing (Definition 16).

Anonymity. Anonymity guarantees that a showing reveals neither the user nor the intermediate issuers in its chain, even to a coalition of fewer than t parties of each honest issuer. In the game, the adversary chooses the root public key and interacts with the challenger, who plays honest intermediate issuers and users; the adversary may hold the shares of up to $t-1$ parties of each honest issuer. It then names two honest users, possibly with different credential chains, as long as all issuers below the root are honest, and wins if it can tell which of the two shows a credential (Definition 15).

4.1 Construction Overview

A credential is a chain of public keys starting at pk_1, in which each key is signed with TMS by the key before it; leveled cross-scheme correctness (Definition 8) is what allows a key to be signed by a key of the level above.

Setup and Keys. Setup generates TMS parameters for each issuer level $k \in [L-1]$: $\mathsf{pp}_{L-1} \leftarrow \mathsf{TMS.Setup}(1^\lambda, 1^2)$ for the issuers that sign user keys, and $\mathsf{pp}_{k-1} \leftarrow \mathsf{TMS.ExtendSetup}(\mathsf{pp}_k)$ for $k = L-1, \ldots, 2$, so that a key generated under pp_k is a message under pp_{k-1}. $\mathsf{IssKeyGen}(\mathsf{pp}, k, \mathsf{pk}')$ runs $\mathsf{TMS.KeyGen}(\mathsf{pp}_k, n, t)$ through a trusted dealer (or, for the root, a DKG). Except for the root ($\mathsf{pk}' = \perp$), the dealer then encodes the new key as a message for pk' by running $\mathsf{TMS.GenTagMsg}$ under pp_{k-1} on the signing key sk_0, which binds the encoding to its signer. It also computes the issuance proof described below, and then erases sk_0 and $\vec{\rho}$. $\mathsf{UserKeyGen}$ samples $\mathsf{sk} = (m_1, m_2) \xleftarrow{\$} (\mathbb{Z}_p^\times)^2$ and sets $\mathsf{pk} = (\hat{P}^{m_1}, \hat{P}^{m_2})$; the user encodes pk for the issuer's key at issuance. User keys never sign.

Issuance. Since the receiver's key, together with its encoding, is a message for the issuer's key, issuance is a threshold signing. One designated receiver party runs the issuance protocol and forwards the result to the others. The issuer must not learn the tag secret $\vec{\rho}$, which would let it recognize the tag in later showings. As in the anonymous credential scheme of [40], the receiver therefore sends its key $\vec{N}$, $R_j = P^{\rho_j}$, the tag $\vec{T}$, and $\vec{M}$, together with a non-interactive zero-knowledge proof of knowledge of $\vec{\rho}$ such that $\log_P R_j = \log_h T_j$ for all j, where $h = \mathsf{H}(\vec{R} \| \vec{N} \| \mathsf{pk})$ and pk is the issuer's key. Each issuer party checks the proof and $e(M_j, \hat{P}) = e(T_j, N_j)$, and rejects if some $R_j = 1$, just as $\mathsf{VerifyMsg}$ rejects tag secrets with a zero entry. It then computes its partial signature as $\mathsf{TMS.ParSign}$ would, using $\vec{R}$ and $\vec{T}$ in place of $\vec{\rho}$ (*e.g.*, $b_i = \prod_j T_j^{z_{i,j}}$), and returns it with its partial public key and the issuer's credential. $\mathsf{CredComb}$ combines the partial signatures with $\mathsf{TMS.Reconst}$ and appends the receiver's key, its encoding $(\vec{T}, \vec{M})$, and the signature to the issuer's credential. UIssue and $\mathsf{UReceive}$ work the same way with a single receiver, the user, who computes the encoding and the proof itself.

Showing. Prove randomizes the entire chain except the root key. For each $k \in [2, L]$, the prover samples $\omega_k, \gamma_k \xleftarrow{\$} \mathbb{Z}_p^\times$ and sets $\mathsf{pk}'_k = \mathsf{TMS.ConvertPK}(\mathsf{pk}_k; \omega_k)$.

For each signature σ_k by pk_{k-1} on pk_k, it runs $\mathsf{TMS.ConvertSig}$ with ω_{k-1} (skipped for the root, $k = 2$), so that the signature verifies under pk'_{k-1}, and then $\mathsf{TMS.ChangeRep}$ with (γ_k, ω_k), which maps the encoding $(\vec{T}_k, \vec{M}_k, \mathsf{pk}_k)$ to $(\vec{T}_k^{\gamma_k}, \vec{M}_k^{\gamma_k \omega_k}, \mathsf{pk}'_k)$. Besides updating the message, $\mathsf{ChangeRep}$ rerandomizes h, which would otherwise identify the signature to its issuer. The prover sends the randomized chain and proves knowledge of the secret key of pk'_L [16]; the verifier checks every signature in the chain with $\mathsf{TMS.Verify}$, which also rejects identity entries in tags and keys, and that the chain starts at pk_1.

References

1. Abe, M., Fuchsbauer, G., Groth, J., Haralambiev, K., Ohkubo, M.: Structure-preserving signatures and commitments to group elements. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 209–236. Springer, Berlin, Heidelberg (Aug 2010)
2. Abe, M., Groth, J., Haralambiev, K., Ohkubo, M.: Optimal structure-preserving signatures in asymmetric bilinear groups. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 649–666. Springer, Berlin, Heidelberg (Aug 2011)
3. Abe, M., Nanri, M., Ohkubo, M., Kempner, O.P., Slamanig, D., Tibouchi, M.: Scalable mixnets from two-party mercurial signatures on randomizable ciphertexts. In: Computer Security – ESORICS 2025 (2025)
4. Abe, M., Nanri, M., Perez-Kempner, O., Tibouchi, M.: Interactive threshold mercurial signatures and applications. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part III. LNCS, vol. 15486, pp. 69–103. Springer, Singapore (Dec 2024)
5. Backes, M., Hanzlik, L., Kluczniak, K., Schneider, J.: Signatures with flexible public key: Introducing equivalence classes for public keys. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part II. LNCS, vol. 11273, pp. 405–434. Springer, Cham (Dec 2018)
6. Bauer, B., Fuchsbauer, G., Regen, F.: On proving equivalence class signatures secure from non-interactive assumptions. In: Tang, Q., Teague, V. (eds.) PKC 2024, Part I. LNCS, vol. 14601, pp. 3–36. Springer, Cham (Apr 2024)
7. Baum, C., Blazy, O., Hoepman, J.H., Lehmann, A., Lysyanskaya, A., Mayrhofer, R., Montgomery, H., Nguyen, N.K., abhi shelat, Slamanig, D., Thomsen, S.E., Camenisch, J., Lee, E., Preneel, B., Tessaro, S., Troncoso, C.: Cryptographers' feedback on the eu digital identity's arf (2024)
8. Belenkiy, M., Camenisch, J., Chase, M., Kohlweiss, M., Lysyanskaya, A., Shacham, H.: Randomizable proofs and delegatable anonymous credentials. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 108–125. Springer, Berlin, Heidelberg (Aug 2009)
9. Bellare, M., Crites, E.C., Komlo, C., Maller, M., Tessaro, S., Zhu, C.: Better than advertised security for non-interactive threshold signatures. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part IV. LNCS, vol. 13510, pp. 517–550. Springer, Cham (Aug 2022)
10. Bellare, M., Goldreich, O.: On defining proofs of knowledge. In: Brickell, E.F. (ed.) CRYPTO'92. LNCS, vol. 740, pp. 390–420. Springer, Berlin, Heidelberg (Aug 1993)
11. Blum, M., Santis, A.D., Micali, S., Persiano, G.: Noninteractive zero-knowledge. SIAM J. Comput. 20(6), 1084–1118 (1991)

12. Bobolz, J., Eidens, F., Krenn, S., Ramacher, S., Samelin, K.: Issuer-hiding attribute-based credentials. In: Conti, M., Stevens, M., Krenn, S. (eds.) CANS 21. LNCS, vol. 13099, pp. 158–178. Springer, Cham (Dec 2021)
13. Camenisch, J., Dubovitskaya, M., Haralambiev, K., Kohlweiss, M.: Composable and modular anonymous credentials: Definitions and practical constructions. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, Part II. LNCS, vol. 9453, pp. 262–288. Springer, Berlin, Heidelberg (Nov / Dec 2015)
14. Camenisch, J., Krenn, S., Lehmann, A., Mikkelsen, G.L., Neven, G., Pedersen, M.Ø.: Formal treatment of privacy-enhancing credential systems. In: Dunkelman, O., Keliher, L. (eds.) SAC 2015. LNCS, vol. 9566, pp. 3–24. Springer, Cham (Aug 2016)
15. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 93–118. Springer, Berlin, Heidelberg (May 2001)
16. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups (extended abstract). In: Kaliski, Jr., B.S. (ed.) CRYPTO'97. LNCS, vol. 1294, pp. 410–424. Springer, Berlin, Heidelberg (Aug 1997)
17. Chase, M., Lysyanskaya, A.: On signatures of knowledge. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 78–96. Springer, Berlin, Heidelberg (Aug 2006)
18. Chaum, D.: Security without identification: Transaction systems to make big brother obsolete. Commun. ACM 28(10), 1030–1044 (1985)
19. Crites, E.C., Kohlweiss, M., Preneel, B., Sedaghat, M., Slamanig, D.: Threshold structure-preserving signatures. In: Guo, J., Steinfeld, R. (eds.) ASIACRYPT 2023, Part II. LNCS, vol. 14439, pp. 348–382. Springer, Singapore (Dec 2023)
20. Crites, E.C., Lysyanskaya, A.: Delegatable anonymous credentials from mercurial signatures. In: Matsui, M. (ed.) CT-RSA 2019. LNCS, vol. 11405, pp. 535–555. Springer, Cham (Mar 2019)
21. Crites, E.C., Lysyanskaya, A.: Mercurial signatures for variable-length messages. PoPETs 2021(4), 441–463 (Oct 2021)
22. Derler, D., Slamanig, D.: Key-homomorphic signatures: definitions and applications to multiparty signatures and non-interactive zero-knowledge. DCC 87(6), 1373–1413 (2019)
23. Desmedt, Y.: Society and group oriented cryptography: A new concept. In: Pomerance, C. (ed.) CRYPTO'87. LNCS, vol. 293, pp. 120–127. Springer, Berlin, Heidelberg (Aug 1988)
24. Desmedt, Y., Frankel, Y.: Threshold cryptosystems. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 307–315. Springer, New York (Aug 1990)
25. Doerner, J., Kondi, Y., Lee, E., shelat, a.: Threshold ECDSA from ECDSA assumptions: The multiparty case. In: 2019 IEEE Symposium on Security and Privacy. pp. 1051–1066. IEEE Computer Society Press (May 2019)
26. Doerner, J., Kondi, Y., Lee, E., Shelat, A., Tyner, L.: Threshold bbs+ signatures for distributed anonymous credential issuance. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 773–789 (2023)
27. Regulation (EU) 2024/1183 of the European Parliament and of the Council of 11 April 2024 amending Regulation (EU) No 910/2014 as regards establishing the European Digital Identity Framework. Official Journal of the European Union (2024)

28. Ernstberger, J., Chaliasos, S., Kadianakis, G., Steinhorst, S., Jovanovic, P., Gervais, A., Livshits, B., Orrù, M.: zk-bench: A toolset for comparative evaluation and performance benchmarking of SNARKs. In: Galdi, C., Phan, D.H. (eds.) SCN 24, Part I. LNCS, vol. 14973, pp. 46–72. Springer, Cham (Sep 2024)
29. Flamini, A., Lee, E., Lysyanskaya, A.: Multi-holder anonymous credentials from BBS signatures. In: Kalai, Y.T., Kamara, S.F. (eds.) CRYPTO 2025, Part VI. LNCS, vol. 16005, pp. 325–357. Springer, Cham (Aug 2025)
30. Fuchsbauer, G., Hanser, C., Slamanig, D.: Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. Journal of Cryptology 32(2), 498–546 (Apr 2019)
31. Gennaro, R., Goldfeder, S., Narayanan, A.: Threshold-optimal DSA/ECDSA signatures and an application to Bitcoin wallet security. In: Manulis, M., Sadeghi, A.R., Schneider, S. (eds.) ACNS 2016. LNCS, vol. 9696, pp. 156–174. Springer, Cham (Jun 2016)
32. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. Journal of Cryptology 20(1), 51–83 (Jan 2007)
33. Ghadafi, E.: Short structure-preserving signatures. In: Sako, K. (ed.) CT-RSA 2016. LNCS, vol. 9610, pp. 305–321. Springer, Cham (Feb / Mar 2016)
34. Griffy, S., Lysyanskaya, A.: PACIFIC: Privacy-preserving automated contact tracing featuring integrity against cloning. CiC 1(2), 12 (2024)
35. Griffy, S., Lysyanskaya, A., Mir, O., Perez-Kempner, O., Slamanig, D.: Delegatable anonymous credentials from mercurial signatures with stronger privacy. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part II. LNCS, vol. 15485, pp. 296–325. Springer, Singapore (Dec 2024)
36. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer, Berlin, Heidelberg (Apr 2008)
37. Hébant, C., Pointcheval, D.: Traceable constant-size multi-authority credentials. In: Galdi, C., Jarecki, S. (eds.) SCN 22. LNCS, vol. 13409, pp. 411–434. Springer, Cham (Sep 2022)
38. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 177–194. Springer, Berlin, Heidelberg (Dec 2010)
39. Lindell, Y., Nof, A.: Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 1837–1854. ACM Press (Oct 2018)
40. Mir, O., Bauer, B., Griffy, S., Lysyanskaya, A., Slamanig, D.: Aggregate signatures with versatile randomization and issuer-hiding multi-authority anonymous credentials. In: Meng, W., Jensen, C.D., Cremers, C., Kirda, E. (eds.) ACM CCS 2023. pp. 30–44. ACM Press (Nov 2023)
41. Mir, O., Slamanig, D., Mayrhofer, R.: Threshold delegatable anonymous credentials with controlled and fine-grained delegation. IEEE Transactions on Dependable and Secure Computing 21(4), 2312–2326 (2024)
42. Mitrokotsa, A., Mukherjee, S., Sedaghat, M., Slamanig, D., Tomy, J.: Threshold structure-preserving signatures: Strong and adaptive security under standard assumptions. In: Tang, Q., Teague, V. (eds.) PKC 2024, Part I. LNCS, vol. 14601, pp. 163–195. Springer, Cham (Apr 2024)
43. Shamir, A.: How to share a secret. Commun. ACM 22(11), 612–613 (1979)

44. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 256–266. Springer, Berlin, Heidelberg (May 1997)
45. Sonnino, A., Al-Bassam, M., Bano, S., Meiklejohn, S., Danezis, G.: Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. In: NDSS 2019. The Internet Society (Feb 2019)

Supplementary Material

A Additional Preliminaries

This section defines the zero-knowledge proofs used by our TDAC construction and the (non-threshold) mercurial signatures to which our threshold reductions reduce.

A.1 Zero-Knowledge Proofs of Knowledge

Our TDAC construction (Appendix E) uses a non-interactive zero-knowledge proof of knowledge [10, 11] NIZK in the random oracle model for issuance (*e.g.*, Fiat–Shamir applied to a Chaum–Pedersen proof), and an interactive zero-knowledge proof of knowledge of a discrete logarithm [16] for showing. For an NP relation $\mathcal{R}$, NIZK consists of two PPT algorithms with access to the random oracle: $\mathsf{P}(\mathsf{stmt}, \mathsf{wit}) \to \pi$ outputs a proof for a statement stmt with witness wit such that $(\mathsf{stmt}, \mathsf{wit}) \in \mathcal{R}$, and $\mathsf{V}(\mathsf{stmt}, \pi) \to \{0, 1\}$ checks a proof. We write $\mathsf{NIZK}\{\mathsf{wit} : (\mathsf{stmt}, \mathsf{wit}) \in \mathcal{R}\}$ for a proof computed by P.

Definition 9 (NIZK proof of knowledge). *NIZK is a NIZK proof of knowledge for $\mathcal{R}$ if it satisfies the following properties.*

- *Completeness. For all* $(\mathsf{stmt}, \mathsf{wit}) \in \mathcal{R}$, $\mathsf{V}(\mathsf{stmt}, \mathsf{P}(\mathsf{stmt}, \mathsf{wit})) = 1$.
- *Zero-knowledge. There is a* PPT *simulator that, given only* stmt *and control of the random oracle, outputs proofs that no* PPT *adversary, which chooses* $(\mathsf{stmt}, \mathsf{wit}) \in \mathcal{R}$ *adaptively, can distinguish from proofs computed by* $\mathsf{P}(\mathsf{stmt}, \mathsf{wit})$*, except with negligible advantage.*
- *Knowledge soundness. There is a* PPT *extractor that, given rewindable black-box access to a prover that outputs an accepting proof for* stmt *with non-negligible probability, and control of the random oracle, outputs* wit *with* $(\mathsf{stmt}, \mathsf{wit}) \in \mathcal{R}$ *with non-negligible probability.*

We require the interactive proof of knowledge used in showing to satisfy the interactive analogues of these properties.

A.2 Non-Threshold Mercurial Signatures

Our threshold reductions (the proofs of Theorems 5 and 6) reduce to the security of $\mathbf{\Pi}_{\mathsf{MBGLS}}$, a non-threshold mercurial signature scheme. Here we write out the syntax of such schemes and the two security games that these reductions use, in the notation of Section 3.

Definition 10 (Mercurial signatures). *A (non-threshold) mercurial signature scheme* MS *consists of the algorithms of Definition 3, except that* $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$ *outputs a single key pair, and* $\mathsf{Sign}(\mathsf{sk}, \mathsf{pk}, \vec{\rho}, (\vec{M}, \vec{N})) \to \sigma$*, which signs a message under the key pair* $(\mathsf{sk}, \mathsf{pk})$*, replaces* $\mathsf{ParSign}$*,* $\mathsf{ParVerify}$*,*

and Reconst. *Correctness, message and tag class-hiding, origin-hiding, and leveled cross-scheme correctness are defined as in Definitions 4 and 6 to 8 with* $n = t = 1$.

Definition 11 (Unforgeability). *A mercurial signature scheme* MS *is existentially unforgeable under chosen tagged message attack (*EUF-CtMA*) if, for all* $\lambda \in \mathbb{N}$ *and all* PPT *adversaries* $\mathcal{A}$, $\Pr\left[\mathsf{EUF\text{-}CtMA}^{\mathsf{MS}}_{\mathcal{A}}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$, *where the game is defined in Figure 5.*

Definition 12 (Public key class-hiding). *A mercurial signature scheme* MS *is public key class-hiding if, for all* $\lambda \in \mathbb{N}$ *and all* PPT *adversaries* $\mathcal{A}$, $\Pr\left[\mathsf{PK\text{-}CLH}^{\mathsf{MS}}_{\mathcal{A}}(1^\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$, *where the game is defined in Figure 5.*

These games are those of Figures 1 and 2 with $n = t = 1$, for which $\mathcal{C} = \emptyset$, the only partial key is the global key, ParSign is Sign, and Reconst returns its single input. For $\mathbf{\Pi}_{\mathsf{MBGLS}}$, ConvertSig keeps h and raises b and s to the power ω, so the oracle $\mathcal{O}_{\mathsf{Sign}^1_2}$ returns $\mathsf{Sign}(\omega\mathsf{sk}_1, \mathsf{pk}^1_2, \vec{\rho}, (\vec{M}, \vec{N}))$, a signature under the converted secret key. Without the boxed elements, the public key class-hiding game is therefore the single-signer version of the game of [40], in which aux contains only the signer's key (Section 3.2). The unforgeability game of [40] asks for a tag secret with the forgery and accepts a forgery whose message class or tag class is new. Ours asks for no tag secret, accepts forgeries under any key in $[\mathsf{pk}]_{\mathcal{R}_\mathcal{K}}$, and requires a new class of $\vec{N}^*$ (see the remark after Definition 5).

B Generalization of $\mathbf{\Pi}_{\mathsf{TMS}}$ Construction to ℓ

In this section, we give our threshold mercurial signature scheme for messages of arbitrary length $\ell \in \mathsf{poly}(\lambda)$, which we call $\mathbf{\Pi}^{\ell}_{\mathsf{TMS}}$, and state its security.

Equivalence classes. For every ℓ, the equivalence classes of messages, tags, and public keys are

$$[(\vec{M}, \vec{N})]_{\mathcal{R}_\mathcal{M}} = \{(\vec{M}^\mu, \vec{N}^\nu) : \mu, \nu \in \mathbb{Z}_p^\times\}, \qquad [\vec{T}]_{\mathcal{R}_\mathcal{T}} = \{\vec{T}^\gamma : \gamma \in \mathbb{Z}_p^\times\},$$
$$[\mathsf{pk}]_{\mathcal{R}_\mathcal{K}} = \{\mathsf{pk}^\omega : \omega \in \mathbb{Z}_p^\times\},$$

where exponentiation is entrywise. ChangeRep with (μ, ν) maps a message $(\vec{M}, \vec{N})$ to $(\vec{M}^{\mu\nu}, \vec{N}^\nu) \in [(\vec{M}, \vec{N})]_{\mathcal{R}_\mathcal{M}}$ and its tag $\vec{T}$ to $\vec{T}^\mu \in [\vec{T}]_{\mathcal{R}_\mathcal{T}}$, and ConvertPK with ω maps pk to $\mathsf{pk}^\omega \in [\mathsf{pk}]_{\mathcal{R}_\mathcal{K}}$. Unforgeability compares the $\mathbb{G}_2$-parts of messages by the same relation, $[\vec{N}]_{\mathcal{R}_\mathcal{K}} = \{\vec{N}^\nu : \nu \in \mathbb{Z}_p^\times\}$. Partial keys are never converted. In the cross-scheme correct version, the encoding $(\vec{T}^{(\mathsf{pk})}, (\vec{M}^{(\mathsf{pk})}, \mathsf{pk}))$ of a key pk as a message for a key pk' is a tagged message under $\mathsf{pp}' = \mathsf{ExtendSetup}(\mathsf{pp})$, and it is randomized like any other: $\mathsf{ChangeRep}_{\mathsf{pp}'}$ with (γ, ω) maps it to $((\vec{T}^{(\mathsf{pk})})^\gamma, ((\vec{M}^{(\mathsf{pk})})^{\gamma\omega}, \mathsf{pk}^\omega))$, the encoding of $\mathsf{ConvertPK}(\mathsf{pk}; \omega)$.

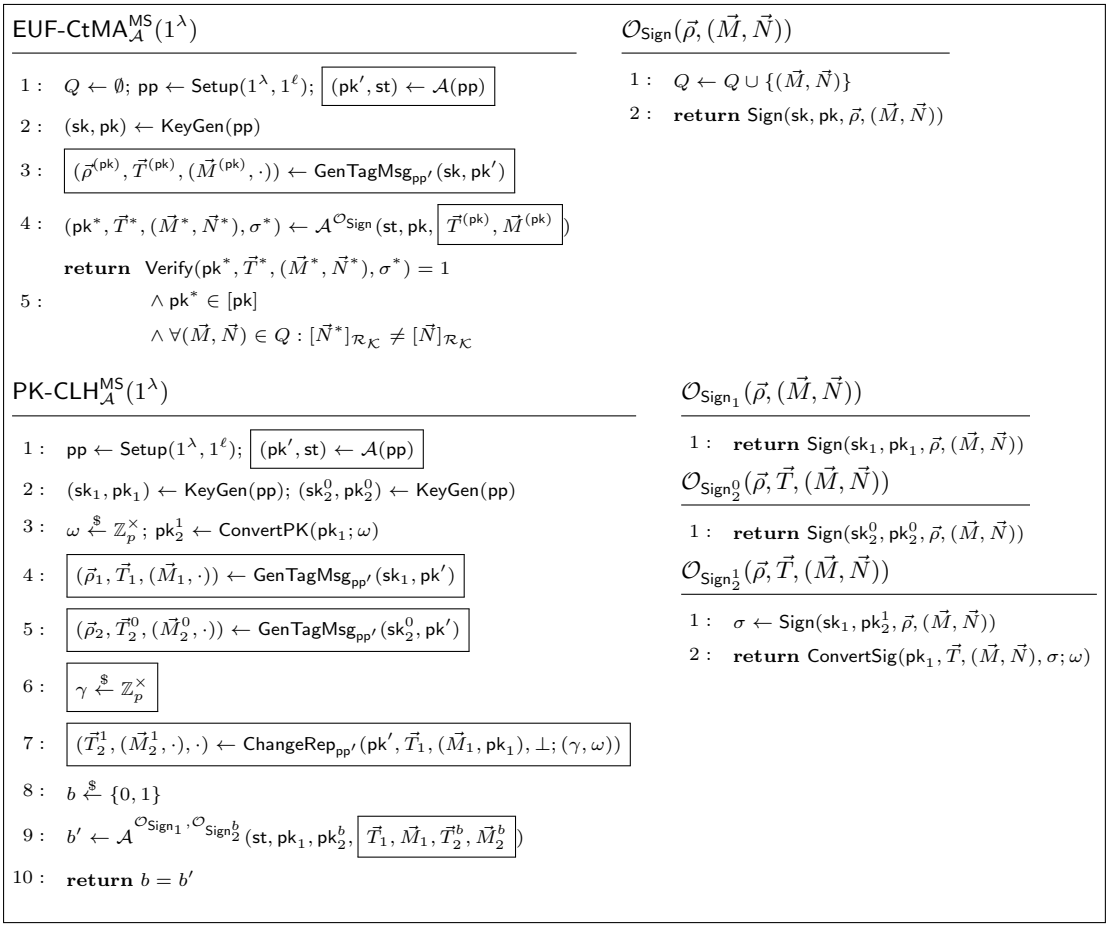

Fig. 5: Unforgeability and public key class-hiding games for a (non-threshold) mercurial signature scheme MS. Boxed elements belong only to the cross-scheme correct version; $\mathsf{GenTagMsg}_{\mathsf{pp}'}$ and $\mathsf{ChangeRep}_{\mathsf{pp}'}$ run with $\mathsf{pp}' = \mathsf{ExtendSetup}(\mathsf{pp})$, as in Figures 1 and 2.

B.1 Construction of TMS for arbitrary ℓ

Figure 6 presents $\mathbf{\Pi}^\ell_{\mathsf{TMS}}$. It is Figure 4 with $\mathbf{\Pi}_{\mathsf{MBGLS}}$ for messages of length ℓ in place of $\mathbf{\Pi}_{\mathsf{MBGLS}}$ for $\ell = 2$ (Figure 3): secret keys have $2\ell + 1$ components, $\mathsf{sk}_0 = (x, y_1, \ldots, y_\ell, z_1, \ldots, z_\ell)$, while messages, tags, and tag secrets have ℓ components. For $n = t = 1$, KeyGen outputs $\mathsf{sk}_1 = \mathsf{sk}_0$ and $\mathsf{pk}_1 = \mathsf{pk}_0$, and ParSign is the signing algorithm Sign of $\mathbf{\Pi}_{\mathsf{MBGLS}}$ for general ℓ. In the cross-scheme correct version, a key pk generated under pp is signed under $\mathsf{pp}' = \mathsf{ExtendSetup}(\mathsf{pp})$, whose message length is $2\ell + 1$, as the tagged message output by $\mathsf{GenTagMsg}_{\mathsf{pp}'}(\mathsf{sk}_0, \mathsf{pk}')$, whose $\mathbb{G}_2$-part is $\hat{P}^{\mathsf{sk}_0} = \mathsf{pk}$.

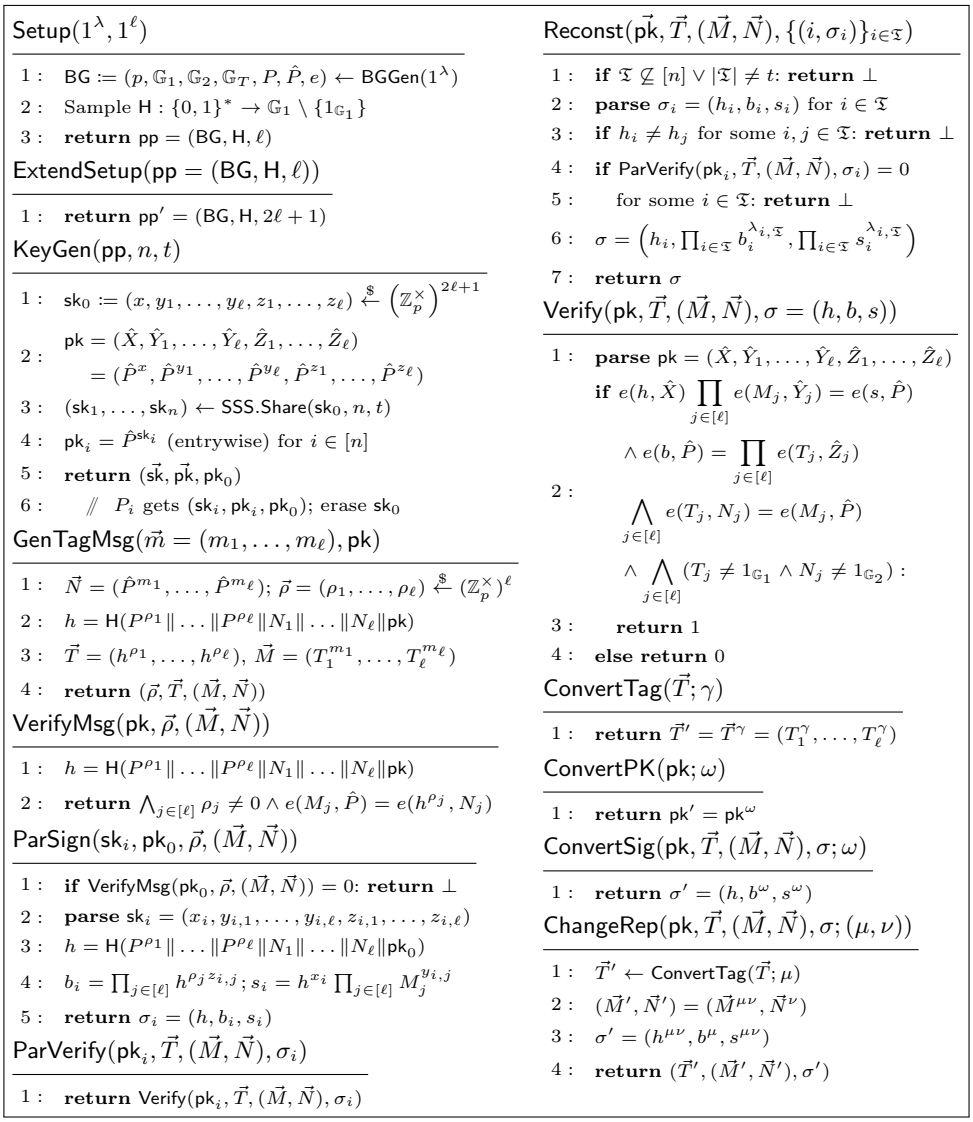

Fig. 6: Threshold mercurial signature scheme $\mathbf{\Pi}^\ell_{\mathsf{TMS}}$ for messages of length ℓ. All algorithms take pp as an implicit input. For $\ell = 2$, this is Figure 4 with the algorithms of Figure 3; for $n = t = 1$, it is $\mathbf{\Pi}_{\mathsf{MBGLS}}$ for messages of length ℓ, with $\mathsf{Sign} = \mathsf{ParSign}$.

B.2 Security of $\mathbf{\Pi}^{\ell}_{\mathsf{TMS}}$

Theorem 7. *Let $\ell \in \mathsf{poly}(\lambda)$. Theorems 1 to 3 hold for $\mathbf{\Pi}_{\mathsf{MBGLS}}$ for messages of length ℓ, and Theorems 4 to 6 and Corollary 1 hold for $\mathbf{\Pi}^{\ell}_{\mathsf{TMS}}$ with respect to it, in each case for both versions.*

Proof (sketch). The proofs for $\ell = 2$ use ℓ only through index ranges. Correctness is the computation in the proof of Theorem 4 with [2] replaced by $[\ell]$, and leveled cross-scheme correctness is shown there for general ℓ. Messages, tags, and their randomization are those of [40] for length ℓ and act entrywise, so class-hiding and origin-hiding follow as for $\ell = 2$. The thresholdization reductions in the proofs of Theorems 5 and 6 use only the entrywise Shamir sharing of the signing key and the fact that all partial signatures on a message share h, so they apply verbatim. The generic group proofs in Appendices C and D.2 are written for $\ell = 2$, where keys and key encodings have 5 components. For general ℓ, message and tag indices range over $[\ell]$ and key indices over $[2\ell+1]$, and each step of these proofs argues index by index. □

C Unforgeability Proof for $\mathbf{\Pi}_{\mathsf{MBGLS}}$

In this section, we prove Theorem 2, the unforgeability (Definition 11) of both versions of $\mathbf{\Pi}_{\mathsf{MBGLS}}$ (Figure 3), in the GGM and ROM. Our unforgeability game differs from that of [40] (see the remark after Definition 12), so we prove it directly, also for the plain version. We write the proof for the cross-scheme correct version. In the plain version, $\mathcal{A}$ does not receive the key encoding $(\vec{T}, \vec{M})$, and the same proof applies with all scalars τ_i and μ_i below set to 0.

The identity checks in Verify are needed in the cross-scheme correct version. Let $(\vec{T}, \vec{M})$ be the key encoding of pk (Section 3.2). Without the checks, $(T_1, 1, M_1)$ would be a valid signature for the tag $(1, 1)$ and every message $((1, 1), \vec{N}')$, and (T_1, M_4M_5, M_1) would be a valid signature for the tag (T_4, T_5) and the message $((1, 1), (1, 1))$. In DAC, the latter would certify the identity key. In the proof below, the checks ensure that the discrete logs of $\vec{T}^*$ and $\vec{N}^*$ are nonzero polynomials.

Proof (of Theorem 2 (Unforgeability of $\mathbf{\Pi}_{\mathsf{MBGLS}}$)). We will prove Theorem 2 in the generic group model (GGM) and random oracle model (ROM). In addition to the signing oracle $\mathcal{O}^{\mathsf{Sign}}$ and hash oracle $\mathcal{O}^{H}$ (which we model as a random oracle), the adversary $\mathcal{A}$ will also have access to the GGM oracles $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$, $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$, and $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_T}$. We will define (or redefine) these oracles below.

Redefining the Game. After $\mathcal{A}$ chooses its key pk' under $\mathsf{pp}' = \mathsf{ExtendSetup}(\mathsf{pp})$, the challenger $\mathcal{B}$ generates random encodings for the secret keys x, y_1, y_2, z_1, z_2 and tag secrets $\rho_1, \rho_2, \rho_3, \rho_4, \rho_5$, as well as the group elements $\mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_5) = (\hat{X}, \hat{Y}_1, \hat{Y}_2, \hat{Z}_1, \hat{Z}_2) = (\hat{P}^x, \hat{P}^{y_1}, \hat{P}^{y_2}, \hat{P}^{z_1}, \hat{P}^{z_2})$, $h = H(P^{\rho_1}||P^{\rho_2}||P^{\rho_3}||P^{\rho_4}||P^{\rho_5}||\mathsf{pk}||\mathsf{pk}')$, $\vec{T} = (T_i)_{i\in[5]} = (h^{\rho_i})_{i\in[5]}$, and $\vec{M} =$

$(T_1^x, T_2^{y_1}, T_3^{y_2}, T_4^{z_1}, T_5^{z_2})$. Here $(\vec{T}, \vec{M})$ is the encoding of pk as a message for pk'; we write η for the discrete log of h. $\mathcal{B}$ stores all these encodings and sends them to $\mathcal{A}$. As in Appendix D.2, $\mathcal{A}$ can query h to $\mathcal{O}^H$ only with negligible probability, since it never sees P^{ρ_i}.

$\mathcal{A}$ can make queries to a number of oracles. The primary ones are GGM oracles for elements of $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$. Each of these three oracles, which we denote $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$, $\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}$, and $\mathcal{O}_{\mathbb{G}_T}^{\mathsf{GGM}}$, takes in a set of scalars and outputs an encoding. Then, $\mathcal{A}$ also has access to the random oracle $\mathcal{O}^H : \{0,1\}^* \to \mathbb{G}_1 \setminus \{1_{\mathbb{G}_1}\}$, which on a fresh input c_i outputs a fresh encoding of some group element P^{η_i}. Finally, $\mathcal{A}$ can make queries to the signature oracle $\mathcal{O}^{\mathsf{Sign}}$, which on input $(\vec{M}^{(k)}, \vec{N}^{(k)}, \tau^{(k)} = (\rho_1^{(k)}, \rho_2^{(k)}))$ computes $h^{(k)} = H(P^{\rho_1^{(k)}} \| P^{\rho_2^{(k)}} \| \hat{N}_1^{(k)} \| \hat{N}_2^{(k)} \| \mathsf{pk})$, returns $\perp$ unless $\rho_j^{(k)} \neq 0$ and $e((h^{(k)})^{\rho_j^{(k)}}, \hat{N}_j^{(k)}) = e(M_j^{(k)}, \hat{P})$ for $j \in [2]$, and otherwise returns $\sigma^{(k)} = (h^{(k)}, b^{(k)}, s^{(k)}) = (h^{(k)}, \prod_{j\in[2]} (h^{(k)})^{\rho_j^{(k)} \mathsf{sk}_{3+j}}, (h^{(k)})^{\mathsf{sk}_1} \prod_{j\in[2]} (M_j^{(k)})^{\mathsf{sk}_{1+j}})$, where $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_5) = (x, y_1, y_2, z_1, z_2)$. This is the oracle of Appendix D.2 with the single key pk, and as there, we assume without loss of generality that $\mathcal{A}$ never repeats a $\mathcal{O}^{\mathsf{Sign}}$ query, so that the discrete logs $\eta^{(k)}$ of the $h^{(k)}$ are distinct indeterminants.

More explicitly, $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$ and $\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}$ are defined as follows:

$$\begin{aligned}
\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}(\alpha, \mu_1, \ldots, \mu_5,\ & \tau_1, \ldots, \tau_5, \gamma_1, \ldots, \gamma_{q_h}, \beta_1, \ldots, \beta_{q_\sigma}, \zeta_1, \ldots, \zeta_{q_\sigma}) \\
&= P^\alpha \prod_{i\in[5]} M_i^{\mu_i} T_i^{\tau_i} \prod_{i\in[q_h]} h_i^{\gamma_i} \prod_{k\in[q_\sigma]} (b^{(k)})^{\beta_k} (s^{(k)})^{\zeta_k} \\
\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}(\alpha, \nu_1, \ldots, \nu_5) &= \hat{P}^\alpha \prod_{i\in[5]} \mathsf{pk}_i^{\nu_i}
\end{aligned}$$

Here q_h represents the number of queries of $\mathcal{O}^H$, and q_σ represents the number of queries of $\mathcal{O}^{\mathsf{Sign}}$. Both of these will increase over the course of the game, adding to the definition of $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$. Also, for a queried signature $\sigma^{(k)} = (h^{(k)}, b^{(k)}, s^{(k)})$, we'll assume for simplicity of notation that $\mathcal{A}$ has already received $h^{(k)}$ as the result of a query of $\mathcal{O}^H$. This way, we don't need separate coefficients for each $h^{(k)}$, but can use the coefficients γ_i.

Note that, in our proof, we will focus on the discrete logs of group elements returned by GGM oracles. These discrete logs are polynomials in $\mathbb{Z}_p[\vec{\kappa}]$, with indeterminants

$$\vec{\kappa} = (\eta, \rho_1, \ldots, \rho_5, \mathsf{sk}_1, \ldots, \mathsf{sk}_5, \eta_1, \ldots, \eta_{q_h})$$

As $\mathcal{O}^H$ is queried more over the course of the game, new encodings h_i are generated, and new indeterminants η_i are added to $\vec{\kappa}$. Now that we've defined our indeterminants, we can write discrete logs of queries of $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$ and $\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}$ as

polynomials of these indeterminants:

$$\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}(\alpha, \mu_1, \ldots, \mu_5,\ \tau_1, \ldots, \tau_5, \gamma_1, \ldots, \gamma_{q_h}, \beta_1, \ldots, \beta_{q_\sigma}, \zeta_1, \ldots, \zeta_{q_\sigma})$$
$$= \alpha + \sum_{i\in[5]} (\mu_i(\eta\rho_i \mathsf{sk}_i) + \tau_i(\eta\rho_i)) + \sum_{i\in[q_h]} \gamma_i(\eta_i) + \sum_{k\in[q_\sigma]} \left(\beta_k \left(\sum_{j\in[2]} \eta^{(k)} \rho_j^{(k)} \mathsf{sk}_{3+j} \right) \right.$$
$$\left. + \zeta_k \left(\eta^{(k)} \mathsf{sk}_1 + \sum_{j\in[2]} p_{M_j^{(k)}}(\vec{\kappa}) \mathsf{sk}_{1+j} \right) \right)$$
$$\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}(\alpha, \nu_1, \ldots, \nu_5) = \alpha + \sum_{i\in[5]} \nu_i(\mathsf{sk}_i)$$

where $p_{T_j^{(k)}}(\vec{\kappa})$ and $p_{M_j^{(k)}}(\vec{\kappa})$ are the discrete log polynomials of $T_j^{(k)}$ and $M_j^{(k)}$, which are the results of some $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$ queries made by $\mathcal{A}$ that were previously passed to the signature oracle. These queries can themselves only be made up of existing indeterminants $\vec{\kappa}$, so each new signature query only adds at most one new indeterminant, that being $\eta^{(k)}$. (Note that $\eta^{(k)} = \eta_i$ for some $i \in [q_h]$).

At any point when a GGM query is made, $\mathcal{B}$ computes the discrete log polynomial of this query. If the polynomial matches a previous query made, $\mathcal{B}$ returns the same encoding it returned for that previous query. Otherwise, it generates and returns a fresh encoding to $\mathcal{A}$.

Finally, once $\mathcal{A}$ has finished querying the oracles, it returns its forgery $(\mathsf{pk}^*, \vec{T}^*, (\vec{M}^*, \vec{N}^*), \sigma^*)$, where $|\mathsf{pk}^*| = 5$ and $|\vec{T}^*| = |\vec{M}^*| = |\vec{N}^*| = 2$, and each element is given as an encoding (that has been previously generated by $\mathcal{B}$). $\mathcal{B}$ takes the discrete log polynomials corresponding to each encoding, and verifies the forgery in indeterminant form. In other words, the polynomials themselves must pass the following conditions, which are the discrete logs of the three verification conditions under pk^*:

$$p_{h^*}(\vec{\kappa}) p_{\mathsf{pk}_1^*}(\vec{\kappa}) + \sum_{j\in[2]} p_{M_j^*}(\vec{\kappa}) p_{\mathsf{pk}_{1+j}^*}(\vec{\kappa}) = p_{s^*}(\vec{\kappa}) \tag{1}$$

$$p_{b^*}(\vec{\kappa}) = \sum_{j\in[2]} p_{T_j^*}(\vec{\kappa}) p_{\mathsf{pk}_{3+j}^*}(\vec{\kappa}) \tag{2}$$

$$p_{T_j^*}(\vec{\kappa}) p_{N_j^*}(\vec{\kappa}) = p_{M_j^*}(\vec{\kappa}) \quad \text{for } j \in [2] \tag{3}$$

Moreover, T_j^* and N_j^* must not be the identity, *i.e.*, $p_{T_j^*}(\vec{\kappa}) \neq 0$ and $p_{N_j^*}(\vec{\kappa}) \neq 0$ for $j \in [2]$. Finally, pk^* must be in $[\mathsf{pk}]_{\mathcal{R}_\mathcal{K}}$, and $\vec{N}^*$ must not be in the class $[\vec{N}^{(k)}]_{\mathcal{R}_\mathcal{K}}$ of the $\mathbb{G}_2$-part of any queried message, where the challenger checks these two conditions for uniformly random values of the indeterminants, sampled at the end of the game. If these conditions hold, $\mathcal{A}$ wins the game. As in Claims 1 and 2, by the Schwartz-Zippel lemma, this generic game is equivalent to the unforgeability game for a generic adversary, except with negligible probability.

Forgery Key. The condition $\mathsf{pk}^* \in [\mathsf{pk}]_{\mathcal{R}_\mathcal{K}}$ means that there is $\omega' \in \mathbb{Z}_p^\times$ such that $p_{\mathsf{pk}_i^*}(\vec{\kappa}) = \omega' \mathsf{sk}_i$ holds at the sampled values for all $i \in [5]$. Then the poly-

nomials $p_{\mathsf{pk}_i^*}(\vec{\kappa})\mathsf{sk}_j - p_{\mathsf{pk}_j^*}(\vec{\kappa})\mathsf{sk}_i$ vanish at these values for all $i, j \in [5]$. They are determined by $\mathcal{A}$'s forgery before the values are sampled, so by the Schwartz-Zippel lemma, they are zero polynomials, except with negligible probability. By definition of $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$, $p_{\mathsf{pk}_i^*}(\vec{\kappa}) = \alpha_i + \sum_{m\in[5]} \nu_{i,m}\mathsf{sk}_m$. For $i \neq j$, the terms $\alpha_i\mathsf{sk}_j$ and $\nu_{i,m}\mathsf{sk}_m\mathsf{sk}_j$ for $m \neq i$ of $p_{\mathsf{pk}_i^*}(\vec{\kappa})\mathsf{sk}_j$ do not occur in $p_{\mathsf{pk}_j^*}(\vec{\kappa})\mathsf{sk}_i$, so $\alpha_i = 0$ and $\nu_{i,m} = 0$ for $m \neq i$. Comparing the terms $\mathsf{sk}_i\mathsf{sk}_j$ then gives $\nu_{i,i} = \nu_{j,j}$. Thus $p_{\mathsf{pk}_i^*}(\vec{\kappa}) = \omega'\mathsf{sk}_i$ for a constant ω', which is nonzero since $\mathsf{pk}^* \in [\mathsf{pk}]_{\mathcal{R}_\mathcal{K}}$. With this, Equations (1) and (2) become

$$\omega' p_{h^*}(\vec{\kappa})\mathsf{sk}_1 + \omega' \sum_{j\in[2]} p_{M_j^*}(\vec{\kappa})\mathsf{sk}_{1+j} = p_{s^*}(\vec{\kappa}) \tag{4}$$

$$p_{b^*}(\vec{\kappa}) = \omega' \sum_{j\in[2]} p_{T_j^*}(\vec{\kappa})\mathsf{sk}_{3+j} \tag{5}$$

Proof of Unforgeability. We call a forgery *valid* if it satisfies all of the above conditions except the class condition, that is, Equations (1) to (3), $p_{T_j^*}(\vec{\kappa}) \neq 0$ and $p_{N_j^*}(\vec{\kappa}) \neq 0$ for $j \in [2]$, and $\mathsf{pk}^* \in [\mathsf{pk}]_{\mathcal{R}_\mathcal{K}}$. We now show that a valid forgery fails the class condition. We first prove that $\mathcal{A}$ must know the discrete log of every $\vec{N}^{(k)}$ that it queries to $\mathcal{O}^{\mathsf{Sign}}$, which determines the discrete logs of the outputs of $\mathcal{O}^{\mathsf{Sign}}$ (Lemma 1). This gives the form of every discrete log in $\mathbb{G}_1$. We then prove that the tag $\vec{T}^*$ of the forgery is made up of the tags of queried messages and of the key encoding (Lemma 2), that it does not contain the key encoding (Lemma 3), and finally that $\vec{N}^*$ is then in the class of some queried $\vec{N}^{(k)}$ (Lemma 4).

Lemma 1. *If $(\vec{M}^{(k)}, \vec{N}^{(k)}, \tau^{(k)})$ is the input to the kth successful $\mathcal{O}^{\mathsf{Sign}}$ query, then there are constants $n_1^{(k)}, n_2^{(k)} \in \mathbb{Z}_p$ such that $p_{N_j^{(k)}}(\vec{\kappa}) = n_j^{(k)}$ and $p_{M_j^{(k)}}(\vec{\kappa}) = \rho_j^{(k)} n_j^{(k)} \eta^{(k)}$ for $j \in [2]$. Hence the discrete logs of $b^{(k)}$ and $s^{(k)}$ are $\eta^{(k)} B_k$ and $\eta^{(k)} S_k$, where*

$$B_k = \sum_{j\in[2]} \rho_j^{(k)}\mathsf{sk}_{3+j}, \qquad S_k = \mathsf{sk}_1 + \sum_{j\in[2]} \rho_j^{(k)} n_j^{(k)}\mathsf{sk}_{1+j}.$$

Proof. We proceed by induction over k, and assume that the lemma holds for all successful $\mathcal{O}^{\mathsf{Sign}}$ queries before the kth. Then, by definition of $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$, the discrete log of every element of $\mathbb{G}_1$ that $\mathcal{A}$ has obtained before the kth query is a sum of a constant, multiples of $\eta\rho_i$, $\eta\rho_i\mathsf{sk}_i$, and of the hash indeterminants η_i, and multiples of $\eta^{(k')}B_{k'}$ and $\eta^{(k')}S_{k'}$ for $k' < k$. Since $\mathcal{A}$ never repeats a $\mathcal{O}^{\mathsf{Sign}}$ query, $\eta^{(k)}$ is distinct from $\eta^{(k')}$ for $k' < k$, and it is distinct from η, since the key encoding hashes a string of a different form. So the only term of such a discrete log that contains $\eta^{(k)}$ is $\gamma_i\eta^{(k)}$, for the index i with $\eta_i = \eta^{(k)}$.

For the kth query to be successful, its input must satisfy $e((h^{(k)})^{\rho_j^{(k)}}, \hat{N}_j^{(k)}) = e(M_j^{(k)}, \hat{P})$ for $j \in [2]$, which has discrete log

$$\rho_j^{(k)}\eta^{(k)} p_{N_j^{(k)}}(\vec{\kappa}) = p_{M_j^{(k)}}(\vec{\kappa}).$$

By definition of $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$, $p_{N_j^{(k)}}(\vec{\kappa}) = \alpha + \sum_{i\in[5]} \nu_i \mathsf{sk}_i$ for some scalars α and ν_i. Every term on the LHS contains $\eta^{(k)}$, and the only such term on the RHS is $\gamma_i\eta^{(k)}$. Since $\rho_j^{(k)} \neq 0$ and the RHS contains no term $\eta^{(k)}\mathsf{sk}_i$, all $\nu_i = 0$. Thus $p_{N_j^{(k)}}(\vec{\kappa})$ is a constant $n_j^{(k)}$, and $p_{M_j^{(k)}}(\vec{\kappa}) = \rho_j^{(k)} n_j^{(k)} \eta^{(k)}$. The discrete logs of $b^{(k)}$ and $s^{(k)}$ then follow from the definition of $\mathcal{O}^{\mathsf{Sign}}$. This concludes our proof of Lemma 1. □

By Lemma 1 and the definition of $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$, the discrete log of every element of $\mathbb{G}_1$ that $\mathcal{A}$ obtains has the form

$$p(\vec{\kappa}) = \alpha + \eta \sum_{i\in[5]} (\tau_i\rho_i + \mu_i\rho_i\mathsf{sk}_i) + \sum_{i\in U} \gamma_i\eta_i + \sum_{k\in[q_\sigma]} \eta^{(k)} \left(\gamma_k + \beta_k B_k + \zeta_k S_k\right), \quad (6)$$

where $U \subseteq [q_h]$ contains the indices of the hashes that are not output by $\mathcal{O}^{\mathsf{Sign}}$, and γ_k is the scalar of $h^{(k)}$. We call η, the η_i for $i \in U$, and the $\eta^{(k)}$ the hash indeterminants. Each term of Equation (6) contains at most one hash indeterminant, and the discrete logs of elements of $\mathbb{G}_2$ contain none. Hence, in each of the polynomial identities above, the terms that contain a given hash indeterminant must agree on both sides, and so must the terms that contain none.

Lemma 2. *If $\mathcal{A}$'s forgery $(\mathsf{pk}^*, \vec{T}^*, (\vec{M}^*, \vec{N}^*), \sigma^*)$ is valid, then there are constants $c_1, c_2 \in \mathbb{Z}_p$ and $t_k \in \mathbb{Z}_p$ for $k \in [q_\sigma]$ such that, for $j \in [2]$,*

$$p_{T_j^*}(\vec{\kappa}) = c_j\eta\rho_{3+j} + \sum_{k\in[q_\sigma]} t_k\rho_j^{(k)}\eta^{(k)}.$$

Proof. By Equation (6), we can write the discrete log polynomial of T_j^* as

$$p_{T_j^*}(\vec{\kappa}) = \alpha^{(j)} + \eta \sum_{i\in[5]} \left(\tau_i^{(j)}\rho_i + \mu_i^{(j)}\rho_i\mathsf{sk}_i\right) + \sum_{i\in U} \gamma_i^{(j)}\eta_i + \sum_{k\in[q_\sigma]} \eta^{(k)} \left(\gamma_k^{(j)} + \beta_k^{(j)} B_k + \zeta_k^{(j)} S_k\right),$$

and that of b^* in the same way, with scalars $\alpha^{(b)}$, $\tau_i^{(b)}$, and so on. We compare the two sides of Equation (5), whose RHS is $\omega' \sum_{j\in[2]} p_{T_j^*}(\vec{\kappa})\mathsf{sk}_{3+j}$.

1. The terms without hash indeterminants are $\alpha^{(b)}$ on the LHS and $\omega' \sum_{j\in[2]} \alpha^{(j)}\mathsf{sk}_{3+j}$ on the RHS, so $\alpha^{(j)} = 0$. In the same way, comparing the terms with η_i for $i \in U$ gives $\gamma_i^{(j)} = 0$.
2. The terms with η on the RHS are $\omega'\eta \sum_{j\in[2]} \sum_{i\in[5]} (\tau_i^{(j)}\rho_i\mathsf{sk}_{3+j} + \mu_i^{(j)}\rho_i\mathsf{sk}_i\mathsf{sk}_{3+j})$. Those on the LHS are $\eta \sum_{i\in[5]} (\tau_i^{(b)}\rho_i + \mu_i^{(b)}\rho_i\mathsf{sk}_i)$, which contain no product of two secret keys, so $\mu_i^{(j)} = 0$. Moreover, $\eta\rho_i\mathsf{sk}_{3+j}$ occurs on the LHS only for $i = 3 + j$, so $\tau_i^{(j)} = 0$ for $i \neq 3 + j$. We let $c_j = \tau_{3+j}^{(j)}$.

3. Fix $k \in [q_\sigma]$. The terms with $\eta^{(k)}$ on the RHS are $\omega'\eta^{(k)} \sum_{j\in[2]}(\gamma_k^{(j)}+\beta_k^{(j)}B_k+\zeta_k^{(j)}S_k)\mathsf{sk}_{3+j}$, and those on the LHS are $\eta^{(k)}(\gamma_k^{(b)} + \beta_k^{(b)}B_k + \zeta_k^{(b)}S_k)$, which contain no product of two secret keys. On the RHS, the term with $\zeta_k^{(j)}$ contains $\eta^{(k)}\mathsf{sk}_1\mathsf{sk}_{3+j}$, which occurs in no other term, so $\zeta_k^{(j)} = 0$. The remaining products of two secret keys on the RHS are $\omega'\eta^{(k)}B_k(\beta_k^{(1)}\mathsf{sk}_4 + \beta_k^{(2)}\mathsf{sk}_5)$. Since $\rho_j^{(k)} \neq 0$, $B_k \neq 0$, so $\beta_k^{(1)} = \beta_k^{(2)} = 0$. Now the RHS has no constant term and no term with sk_1, so $\gamma_k^{(b)} = \zeta_k^{(b)} = 0$, and what remains is $\beta_k^{(b)} \sum_{j\in[2]} \rho_j^{(k)}\mathsf{sk}_{3+j} = \omega' \sum_{j\in[2]} \gamma_k^{(j)}\mathsf{sk}_{3+j}$. Thus $\gamma_k^{(j)} = t_k\rho_j^{(k)}$ for $t_k = \beta_k^{(b)}/\omega'$, which does not depend on j.

Together, this gives the form of $p_{T_j^*}(\vec{\kappa})$ in the lemma. This concludes our proof of Lemma 2. □

Lemma 3. *If $\mathcal{A}$'s forgery $(\mathsf{pk}^*, \vec{T}^*, (\vec{M}^*, \vec{N}^*), \sigma^*)$ is valid, then $c_1 = c_2 = 0$ in Lemma 2. Hence $\vec{T}^*$ and $\vec{M}^*$ are not functions of ρ_i for any $i \in [5]$, and for $j \in [2]$,*

$$p_{T_j^*}(\vec{\kappa}) = \sum_{k\in[q_\sigma]} t_k\rho_j^{(k)}\eta^{(k)}, \qquad p_{M_j^*}(\vec{\kappa}) = p_{N_j^*}(\vec{\kappa}) \sum_{k\in[q_\sigma]} t_k\rho_j^{(k)}\eta^{(k)}.$$

Proof. By Lemma 2 and Equation (3), the terms of $p_{M_j^*}(\vec{\kappa})$ that contain η are $c_j\eta\rho_{3+j}p_{N_j^*}(\vec{\kappa})$. We write the terms of $p_{h^*}(\vec{\kappa})$ that contain η as $\eta\sum_{i\in[5]}(\tau_i^{(h^*)}\rho_i + \mu_i^{(h^*)}\rho_i\mathsf{sk}_i)$, and those of $p_{s^*}(\vec{\kappa})$ in the same way. Then the terms of Equation (4) that contain η are

$$\omega'\eta \sum_{i\in[5]} \left(\tau_i^{(h^*)}\rho_i\mathsf{sk}_1 + \mu_i^{(h^*)}\rho_i\mathsf{sk}_i\mathsf{sk}_1\right) + \omega'\eta \sum_{j\in[2]} c_j\rho_{3+j}p_{N_j^*}(\vec{\kappa})\mathsf{sk}_{1+j}$$
$$= \eta \sum_{i\in[5]} \left(\tau_i^{(s^*)}\rho_i + \mu_i^{(s^*)}\rho_i\mathsf{sk}_i\right).$$

Suppose that $c_j \neq 0$ for some $j \in [2]$. Since $p_{N_j^*}(\vec{\kappa})$ is a nonzero polynomial of the form $\alpha + \sum_{l\in[5]} \nu_l\mathsf{sk}_l$, the term $\omega' c_j\eta\rho_{3+j}p_{N_j^*}(\vec{\kappa})\mathsf{sk}_{1+j}$ is nonzero, and each of its monomials is $\eta\rho_{3+j}\mathsf{sk}_{1+j}$ or $\eta\rho_{3+j}\mathsf{sk}_l\mathsf{sk}_{1+j}$ for some $l \in [5]$. None of these monomials occurs in any other term of the equation. The terms with h^* have monomials $\eta\rho_i\mathsf{sk}_1$ and $\eta\rho_i\mathsf{sk}_i\mathsf{sk}_1$, which differ from them since $1+j \notin \{1, 3+j\}$. The term for the other index $j' \neq j$ contains $\rho_{3+j'}$ instead of ρ_{3+j}. On the RHS, the monomials are $\eta\rho_i$ and $\eta\rho_i\mathsf{sk}_i$, which differ from them since $1 + j \neq 3 + j$. So the equation cannot hold, and $c_j = 0$. The form of $p_{T_j^*}(\vec{\kappa})$ now follows from Lemma 2, and that of $p_{M_j^*}(\vec{\kappa})$ from Equation (3). This concludes our proof of Lemma 3. □

Lemma 4. *If $\mathcal{A}$'s forgery $(\mathsf{pk}^*, \vec{T}^*, (\vec{M}^*, \vec{N}^*), \sigma^*)$ is valid, and $t_k \neq 0$ in Lemma 3 for some $k \in [q_\sigma]$, then there is a constant $c \in \mathbb{Z}_p^\times$ such that $p_{N_j^*}(\vec{\kappa}) = c\, n_j^{(k)}$ for $j \in [2]$.*

Proof. We compare the terms of Equation (4) that contain $\eta^{(k)}$. By Equation (6), we can write these terms in $p_X(\vec{\kappa})$ as $\eta^{(k)}(\gamma^{(X)} + \beta^{(X)}B_k + \zeta^{(X)}S_k)$ for $X \in \{h^*, M_1^*, M_2^*, s^*\}$. Then the terms of Equation (4) that contain $\eta^{(k)}$ are

$$\omega'\eta^{(k)}\left(\gamma^{(h^*)} + \beta^{(h^*)}B_k + \zeta^{(h^*)}S_k\right)\mathsf{sk}_1 + \omega'\eta^{(k)}\sum_{j\in[2]}\left(\gamma^{(M_j^*)} + \beta^{(M_j^*)}B_k + \zeta^{(M_j^*)}S_k\right)\mathsf{sk}_{1+j}$$
$$= \eta^{(k)}\left(\gamma^{(s^*)} + \beta^{(s^*)}B_k + \zeta^{(s^*)}S_k\right).$$

The RHS contains no product of two secret keys, so neither may the LHS. The term with $\zeta^{(h^*)}$ contains $\eta^{(k)}\mathsf{sk}_1^2$, which occurs in no other term, so $\zeta^{(h^*)} = 0$. Then the term with $\zeta^{(M_j^*)}$ contains $\eta^{(k)}\mathsf{sk}_1\mathsf{sk}_{1+j}$, which occurs in no other term, so $\zeta^{(M_j^*)} = 0$. The remaining products of two secret keys are $\omega'\eta^{(k)}B_k(\beta^{(h^*)}\mathsf{sk}_1 + \sum_{j\in[2]}\beta^{(M_j^*)}\mathsf{sk}_{1+j})$, so $\beta^{(h^*)} = \beta^{(M_j^*)} = 0$, since $B_k \neq 0$. Now the LHS has no constant term and no term with sk_4 or sk_5, so $\gamma^{(s^*)} = \beta^{(s^*)} = 0$, and comparing the terms with sk_{1+j} gives $\omega'\gamma^{(M_j^*)} = \zeta^{(s^*)}\rho_j^{(k)}n_j^{(k)}$.

On the other hand, we have just shown that the terms of $p_{M_j^*}(\vec{\kappa})$ that contain $\eta^{(k)}$ are $\gamma^{(M_j^*)}\eta^{(k)}$, and by Lemma 3 they are $t_k\rho_j^{(k)}\eta^{(k)}p_{N_j^*}(\vec{\kappa})$. Hence

$$\omega' t_k \rho_j^{(k)} p_{N_j^*}(\vec{\kappa}) = \omega'\gamma^{(M_j^*)} = \zeta^{(s^*)}\rho_j^{(k)}n_j^{(k)}.$$

Since ω', t_k, and $\rho_j^{(k)}$ are nonzero, $p_{N_j^*}(\vec{\kappa}) = c\, n_j^{(k)}$ for both $j \in [2]$, with $c = \zeta^{(s^*)}/(\omega' t_k)$. Since $p_{N_j^*}(\vec{\kappa}) \neq 0$, also $c \neq 0$. This concludes our proof of Lemma 4. □

Conclusion of Theorem 2. Suppose that $\mathcal{A}$ wins the generic game with forgery $(\mathsf{pk}^*, \vec{T}^*, (\vec{M}^*, \vec{N}^*), \sigma^*)$. Then the forgery is valid, and $p_{\mathsf{pk}_i^*}(\vec{\kappa}) = \omega'\mathsf{sk}_i$ except with negligible probability. Since $p_{T_1^*}(\vec{\kappa}) \neq 0$, Lemma 3 gives $t_k \neq 0$ for some $k \in [q_\sigma]$. By Lemma 4, $\vec{N}^* = (\vec{N}^{(k)})^c$ with $c \in \mathbb{Z}_p^\times$ at all values of the indeterminants, so $[\vec{N}^*]_{\mathcal{R}_\mathcal{K}} = [\vec{N}^{(k)}]_{\mathcal{R}_\mathcal{K}}$, and the forgery fails the class condition. Hence $\mathcal{A}$ wins only with negligible probability, which proves the unforgeability of both versions of $\mathbf{\Pi}_{\mathsf{MBGLS}}$ in the GGM and ROM. □

D PK Class-Hiding Proofs for $\mathbf{\Pi}_{\mathsf{TMS}}$ and $\mathbf{\Pi}_{\mathsf{MBGLS}}$

In this section, we prove Theorems 3 and 6, in that order. We first prove the threshold public key class-hiding (Definition 6) of both the plain and cross-scheme correct versions of $\mathbf{\Pi}_{\mathsf{TMS}}$ (Figure 4), and then the public key class-hiding (Definition 12) of the cross-scheme correct version of $\mathbf{\Pi}_{\mathsf{MBGLS}}$ (Figure 3). For the plain version of $\mathbf{\Pi}_{\mathsf{MBGLS}}$, public key class-hiding was proven in [40].

D.1 Proof of Theorem 6

Recall that we consider two versions of $\mathbf{\Pi}_{\mathsf{TMS}}$: the cross-scheme correct version, whose security games (Figures 1 and 2) contain additional boxed elements, and

the plain version, whose games do not. The former is a strict generalization of the latter, so if we can prove that it is public key class-hiding (PKCH), we are done. However, our proof of PKCH for the cross-scheme correct version of $\mathbf{\Pi}_{\mathsf{TMS}}$ relies on the PKCH of the cross-scheme correct $\mathbf{\Pi}_{\mathsf{MBGLS}}$ (Theorem 3), whose proof is highly non-trivial and requires a lot of setup. Therefore, we will provide proofs for both versions of $\mathbf{\Pi}_{\mathsf{TMS}}$: one for readers interested in the cross-scheme correct version for its use in constructing DAC, and one for readers who only want a working threshold mercurial signature and are not interested in the long proof of Theorem 3. Due to the similar structure of these proofs, we provide both at once, and indicate the differences in $\boxed{\text{solid boxes}}$ when they arise.

Proof (of Theorem 6 (PK class-hiding of $\mathbf{\Pi}_{\mathsf{TMS}}$)). Let $\mathcal{A}$ be a PPT adversary against $\mathsf{PK\text{-}CLH}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{TMS}}}$ (Figure 2). We construct a reduction $\mathcal{B}$ that runs $\mathcal{A}$ and plays $\mathsf{PK\text{-}CLH}_{\mathcal{B}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ (Definition 12) against the plain (resp. cross-scheme correct) version of $\mathbf{\Pi}_{\mathsf{MBGLS}}$ when $\mathcal{A}$ plays against the plain (resp. cross-scheme correct) version of $\mathbf{\Pi}_{\mathsf{TMS}}$. Boxed steps below occur only in the cross-scheme correct case.

Setup. $\mathcal{B}$ receives pp from its challenger and forwards it to $\mathcal{A}$, which returns $(\mathcal{C}, \boxed{\mathsf{pk}'})$. $\mathcal{B}$ sends $\boxed{\mathsf{pk}'}$ to its challenger. Assume without loss of generality that $|\mathcal{C}| = t - 1$.

Key Generation. $\mathcal{B}$ receives $\mathsf{pk}_{1,0}$, $\mathsf{pk}_{2,0}^b$, and $\boxed{\vec{T}_1, \vec{M}_1, \vec{T}_2^b, \vec{M}_2^b}$ from its challenger. For each $i \in \mathcal{C}$, it samples the partial secret keys $\mathsf{sk}_{1,i} \xleftarrow{\$} (\mathbb{Z}_p)^5$ and sets $\mathsf{pk}_{1,i} = \hat{P}^{\mathsf{sk}_{1,i}}$ (entrywise). Then for each honest $k \in \mathcal{H} = [n] \setminus \mathcal{C}$, it generates partial public keys as

$$\mathsf{pk}_{1,k} = \mathsf{pk}_{1,0}^{\lambda_{0,\tilde{\mathcal{T}}}(k)} \prod_{i \in \mathcal{C}} \mathsf{pk}_{1,i}^{\lambda_{i,\tilde{\mathcal{T}}}(k)} \quad \text{(entrywise)},$$

with $\tilde{\mathcal{T}} = \mathcal{C} \cup \{0\}$ and Lagrange coefficients computed as in Section 2. It then sends $\mathsf{pk}_{1,0}$, $\vec{\mathsf{pk}}_1 = (\mathsf{pk}_{1,1}, \ldots, \mathsf{pk}_{1,n})$, $\{\mathsf{sk}_{1,i}\}_{i \in \mathcal{C}}$, $\mathsf{pk}_{2,0}^b$, and $\boxed{\vec{T}_1, \vec{M}_1, \vec{T}_2^b, \vec{M}_2^b}$ to $\mathcal{A}$.

Signing. On a query $\mathcal{O}_{\mathsf{PSign}_1}(i, \vec{\rho}, (\vec{M}, \vec{N}))$ with $i \in [n]$, $\mathcal{B}$ queries its own oracle $\mathcal{O}_{\mathsf{Sign}_1}$ on $(\vec{\rho}, (\vec{M}, \vec{N}))$. If the answer is $\perp$, it returns $\perp$. Otherwise it obtains $\sigma_0 = (h, b_0, s_0)$, computes $\sigma_j = (h, b_j, s_j) \leftarrow \mathsf{ParSign}(\mathsf{sk}_{1,j}, \mathsf{pk}_{1,0}, \vec{\rho}, (\vec{M}, \vec{N}))$ for $j \in \mathcal{C}$ (all with the same h, which depends only on $\vec{\rho}$, $\vec{N}$, and $\mathsf{pk}_{1,0}$), and returns

$$\sigma_i = \Big(h,\ b_0^{\lambda_{0,\tilde{\mathcal{T}}}(i)} \prod_{j \in \mathcal{C}} b_j^{\lambda_{j,\tilde{\mathcal{T}}}(i)},\ s_0^{\lambda_{0,\tilde{\mathcal{T}}}(i)} \prod_{j \in \mathcal{C}} s_j^{\lambda_{j,\tilde{\mathcal{T}}}(i)}\Big).$$

It answers each query to $\mathcal{O}_{\mathsf{Sign}_2^b}$ by forwarding it to its own oracle $\mathcal{O}_{\mathsf{Sign}_2^b}$.

Output. When $\mathcal{A}$ outputs b', $\mathcal{B}$ outputs b'.

Analysis. $\mathcal{B}$ simulates $\mathsf{PK\text{-}CLH}^{\mathbf{\Pi}_{\mathsf{TMS}}}_{\mathcal{A}}$ perfectly, with the challenger's bit b:

- In the real game, the shares of $\mathsf{sk}_{1,0}$ lie on a uniformly random polynomial of degree $t-1$ with constant term $\mathsf{sk}_{1,0}$. Its values at the $t-1$ points in $\mathcal{C}$ are uniform and independent of $\mathsf{sk}_{1,0}$, and together with $\mathsf{sk}_{1,0}$ they determine it, with $\mathsf{sk}_{1,k} = \lambda_{0,\bar{\mathcal{T}}}(k)\mathsf{sk}_{1,0} + \sum_{i\in\mathcal{C}} \lambda_{i,\bar{\mathcal{T}}}(k)\mathsf{sk}_{1,i}$. Hence the corrupted shares and all partial public keys are distributed as in the real game. The keys $\mathsf{pk}_{1,0}$, $\mathsf{pk}^b_{2,0}$, and their encodings come from the challenger, with the same distribution as in the real game.
- The components b and s of $\mathbf{\Pi}_{\mathsf{MBGLS}}.\mathsf{Sign}$ are linear in the signing key for fixed h, $\vec{\rho}$, and $(\vec{M},\vec{N})$, so the formula above yields exactly $\mathsf{ParSign}(\mathsf{sk}_{1,i}, \mathsf{pk}_{1,0}, \vec{\rho}, (\vec{M},\vec{N}))$. Both return $\perp$ exactly when $\mathsf{VerifyMsg}(\mathsf{pk}_{1,0}, \vec{\rho}, (\vec{M},\vec{N})) = 0$.
- In the real game, $\mathcal{O}_{\mathsf{Sign}^b_2}$ reconstructs t partial signatures. By the proof of threshold verification in Theorem 4, $\mathsf{Reconst}$ outputs exactly the $\mathbf{\Pi}_{\mathsf{MBGLS}}$ signature under the reconstructed key, i.e., $\mathsf{Sign}(\mathsf{sk}^0_{2,0}, \mathsf{pk}^0_{2,0}, \vec{\rho}, (\vec{M},\vec{N}))$ for $b = 0$ and $\mathsf{ConvertSig}(\mathsf{pk}_{1,0}, \vec{T}, (\vec{M},\vec{N}), \mathsf{Sign}(\mathsf{sk}_{1,0}, \mathsf{pk}^1_{2,0}, \vec{\rho}, (\vec{M},\vec{N})); \omega)$ for $b = 1$. These are the answers of the oracles of $\mathsf{PK\text{-}CLH}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}_{\mathcal{B}}$.

Hence $\mathcal{B}$ has the same advantage in $\mathsf{PK\text{-}CLH}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}_{\mathcal{B}}$ as $\mathcal{A}$ has in $\mathsf{PK\text{-}CLH}^{\mathbf{\Pi}_{\mathsf{TMS}}}_{\mathcal{A}}$, which proves the theorem. □

D.2 Proof of Theorem 3

Now we provide the proof of Theorem 3, which states the public key class-hiding (Definition 12) of the cross-scheme correct version of $\mathbf{\Pi}_{\mathsf{MBGLS}}$ (Figure 3).

Proof (of Theorem 3 (PK class-hiding of $\mathbf{\Pi}_{\mathsf{MBGLS}}$)). We will prove Theorem 3 in the generic group model (GGM) and random oracle model (ROM). In addition to the signing oracle $\mathcal{O}^{\mathsf{Sign}}$ and hash oracle $\mathcal{O}^{H}$ (which we model as a random oracle), the adversary $\mathcal{A}$ will also have access to the GGM oracles $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$, $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$, and $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_T}$. We will define (or redefine) these oracles below.

Notation. In the game, $\mathcal{A}$ first chooses its key pk' under $\mathsf{pp}' = \mathsf{ExtendSetup}(\mathsf{pp})$. For $\theta \in \{1,2\}$, we write pk_θ for the public key of key θ (so pk_2 is the key pk^b_2 of Figure 5), $\mathsf{pk}_{\theta,i}$ for its ith component, and $(\vec{T}_\theta, \vec{M}_\theta)$ for its encoding as a message for pk'. A fresh key θ is encoded as $(\vec{\rho}_\theta, \vec{T}_\theta, (\vec{M}_\theta, \mathsf{pk}_\theta)) \leftarrow \mathsf{GenTagMsg}_{\mathsf{pp}'}(\mathsf{sk}_\theta, \mathsf{pk}')$, so that $T_{\theta,i} = h_\theta^{\rho_{\theta,i}}$ and $M_{\theta,i} = T_{\theta,i}^{\mathsf{sk}_{\theta,i}}$ for $i \in [5]$, where $h_\theta = H(P^{\rho_{\theta,1}} \| \ldots \| P^{\rho_{\theta,5}} \| \mathsf{pk}_\theta \| \mathsf{pk}')$ has discrete log η_θ. Key 1 is always fresh, and so is key 2 for $b = 0$. For $b = 1$, key 2 is the converted key 1: $\mathsf{pk}_2 = \mathsf{pk}_1^\omega$, $\vec{T}_2 = \vec{T}_1^\psi$, and $\vec{M}_2 = \vec{M}_1^{\psi\omega}$, where ψ is the randomness γ of Figure 5. We write $\mathsf{sk}_1^{(b)} = \mathsf{sk}_1$ and $\mathsf{sk}_2^{(b)}$ for the signing key of key 2, i.e., $\mathsf{sk}_2^{(0)} = \mathsf{sk}_2$ and $\mathsf{sk}_2^{(1)} = \omega\mathsf{sk}_1$, and we add the superscript (b) to the elements of key 2 where the hybrid matters. $\mathcal{A}$ never sees $P^{\rho_{\theta,i}}$, so it can query h_1 or h_2 to $\mathcal{O}^H$ only with negligible probability.

Redefining the Game. The game $\mathsf{PK\text{-}CLH}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}_{\mathcal{A}}$ (Definition 12) can equivalently be formulated in terms of two hybrids in the GGM. Call this game Game 0, and denote the two hybrids by Hybrid 0 and Hybrid 3. The challenger $\mathcal{C}$ of Game 0 begins by sampling $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, $\mathcal{C}$ challenges $\mathcal{A}$ to Hybrid 0, and if $b = 1$, $\mathcal{C}$ challenges $\mathcal{A}$ to Hybrid 3. We define these hybrids as follows:

- **Hybrid 0.** $\mathcal{C}$ sends encodings of $(\vec{T}_1, \vec{M}_1, \mathsf{pk}_1)$ and $(\vec{T}_2^{(0)}, \vec{M}_2^{(0)}, \mathsf{pk}_2^{(0)})$. $\mathcal{A}$ can make queries to $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$, $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$, $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_T}$, $\mathcal{O}^{H}$ and $\mathcal{O}^{\mathsf{Sign}}$. $\mathcal{C}$ computes the query and outputs encoding(s) of the output. $\mathcal{C}$ keeps a table of previous encodings and the values they correspond to, so that if some value comes up again, the same encoding is sent. $\mathcal{A}$ wins the game if its guess $b' = 0$.
- **Hybrid 3.** $\mathcal{C}$ sends encodings of $(\vec{T}_1, \vec{M}_1, \mathsf{pk}_1)$ and $(\vec{T}_2^{(1)}, \vec{M}_2^{(1)}, \mathsf{pk}_2^{(1)})$. $\mathcal{A}$ can query $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$, $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$, $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_T}$, $\mathcal{O}^{H}$ and $\mathcal{O}^{\mathsf{Sign}}$, and $\mathcal{C}$ responds with encodings in the same way as in Hybrid 0. $\mathcal{A}$ wins the game if its guess $b' = 1$.

We can trivially see that Game 0 is equivalent to the $\mathsf{PK\text{-}CLH}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}_{\mathcal{A}}$ game.

Now we introduce Game 1, which is played in the same way but with two different hybrids, Hybrid 1 and Hybrid 2. These hybrids deal only with indeterminants rather than computed values, and are defined as follows:

- **Hybrid 1.** As in Hybrid 0, $\mathcal{C}$ uses the keys $(\vec{T}_1, \vec{M}_1, \mathsf{pk}_1)$ and $(\vec{T}_2^{(0)}, \vec{M}_2^{(0)}, \mathsf{pk}_2^{(0)})$. However, instead of randomly sampling values at the start, $\mathcal{C}$ keeps everything in indeterminant form, where the discrete logs of group elements are formal multivariate polynomials in $\mathbb{Z}_p[\vec{\kappa}]$. $\mathcal{A}$ can query $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$, $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$, $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_T}$, $\mathcal{O}^{H}$, and $\mathcal{O}^{\mathsf{Sign}}$. $\mathcal{C}$ outputs encoding(s) of the discrete log polynomials of the resulting expressions, which are in indeterminant form. $\mathcal{C}$ keeps a table of previous encodings and the formal polynomials they correspond to. As in Hybrid 0, $\mathcal{A}$ wins if its guess $b' = 0$.
- **Hybrid 2.** As in Hybrid 3, $\mathcal{C}$ uses the keys $(\vec{T}_1, \vec{M}_1, \mathsf{pk}_1)$ and $(\vec{T}_2^{(1)}, \vec{M}_2^{(1)}, \mathsf{pk}_2^{(1)})$. But again, no values are actually sampled, and encodings are computed on formal discrete log polynomials. $\mathcal{A}$ has access to $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$, $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$, $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_T}$, $\mathcal{O}^{H}$, and $\mathcal{O}^{\mathsf{Sign}}$, and wins if its guess $b' = 1$.

We make the following claims about these hybrids:

Claim 1 *A generic adversary's view in Hybrid 0 is the same as it is in Hybrid 1.*

Claim 2 *A generic adversary's view in Hybrid 3 is the same as it is in Hybrid 2.*

If these claims are true (and we will prove that they are shortly), then Game 1 is equivalent to Game 0, and thereby also equivalent to $\mathsf{PK\text{-}CLH}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}_{\mathcal{A}}$ in the generic group model. However, before we can prove these claims, we must define how each oracle works.

Defining the Oracles. The adversary $\mathcal{A}$ has access to five oracles:

- $\mathcal{O}^H$ is an oracle for the hash function $H : \{0,1\}^* \to \mathbb{G}_1$. $\mathcal{A}$ can query $\mathcal{O}^H$ with a string s_i and receive an indeterminant h_i that encodes the group element $H(s_i)$.
- $\mathcal{O}^{\mathsf{Sign}}$ is a signature oracle, and functions as follows. On input a key index $\theta_k \in \{1,2\}$ and a message $(\vec{M}^{(k)}, \vec{N}^{(k)}, \tau^{(k)} = (\rho_1^{(k)}, \rho_2^{(k)}))$, compute the hash $h^{(k)} = H(P^{\rho_1^{(k)}} \| P^{\rho_2^{(k)}} \| \hat{N}_1^{(k)} \| \hat{N}_2^{(k)} \| \mathsf{pk}_{\theta_k})$. If $\rho_j^{(k)} \neq 0$ and $e((h^{(k)})^{\rho_j^{(k)}}, \hat{N}_j^{(k)}) = e(M_j^{(k)}, \hat{P})$ for $j \in [2]$, output $\sigma^{(k)} = (h^{(k)}, b^{(k)}, s^{(k)})$, where

$$b^{(k)} = \prod_{j\in[2]} (h^{(k)})^{\rho_j^{(k)} \mathsf{sk}_{\theta_k,3+j}^{(b)}}, \qquad s^{(k)} = (h^{(k)})^{\mathsf{sk}_{\theta_k,1}^{(b)}} \prod_{j\in[2]} (M_j^{(k)})^{\mathsf{sk}_{\theta_k,1+j}^{(b)}}$$

  for $b \in \{0,1\}$ (depending on which hybrid $\mathcal{A}$ is in). Else, output $\perp$. For $\theta_k = 1$ this is $\mathcal{O}_{\mathsf{Sign}_1}$, and for $\theta_k = 2$ it is $\mathcal{O}_{\mathsf{Sign}_2^b}$, which for $b = 1$ signs with $\omega\mathsf{sk}_1$ (see the remark after Definition 12). Since $\mathcal{O}^{\mathsf{Sign}}$ is deterministic, we assume without loss of generality that $\mathcal{A}$ never repeats a query. Then distinct queries hash distinct strings, because $\mathsf{VerifyMsg}$ determines $\vec{M}^{(k)}$ from $\tau^{(k)}$, $\vec{N}^{(k)}$, and pk_{θ_k}, so the discrete logs $\eta^{(k)}$ of the $h^{(k)}$ are distinct indeterminants. We also assume that $\mathcal{A}$ has queried $\mathcal{O}^H$ on the string hashed to each $h^{(k)}$, so that every $h^{(k)}$ is one of the hashes h_i.
- $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$, $\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}$ are GGM oracles. Each oracle takes in a set of scalars and outputs an encoding. Since encodings are sparse random strings, an adversary that submits an encoding it did not receive from an oracle succeeds only with negligible probability. We therefore let the group oracles take exponent vectors over previously received elements, which is equivalent. Explicitly, $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$ and $\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}$ are defined as follows:

$$\begin{aligned}
\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}} &= P^{\alpha} \prod_{i\in[5]} (M_{1,i})^{\mu_{1,i}} (T_{1,i})^{\tau_{1,i}} (M_{2,i}^{(b)})^{\mu_{2,i}} (T_{2,i}^{(b)})^{\tau_{2,i}} \prod_{i\in[q_h]} h_i^{\gamma_i} \\
&\quad \prod_{k\in[q_\sigma]} (b^{(k)})^{\beta_k} (s^{(k)})^{\zeta_k} \\
\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}} &= \hat{P}^{\alpha} \prod_{i\in[5]} (\mathsf{pk}_{1,i})^{\nu_{1,i}} (\mathsf{pk}_{2,i}^{(b)})^{\nu_{2,i}}
\end{aligned}$$

  where b is either 0 or 1 depending on which hybrid we are in. Here $\mathsf{pk}_{\theta,i}$, $T_{\theta,i}$, and $M_{\theta,i}$ are as in the notation paragraph above, and q_h and q_σ are the numbers of $\mathcal{O}^H$ and $\mathcal{O}^{\mathsf{Sign}}$ queries. Note also that, although $\tau^{(k)} = (\rho_1^{(k)}, \rho_2^{(k)})$ is a vector, $\tau_{1,i}$ and $\tau_{2,i}$ are scalars for $i \in [5]$.
- $\mathcal{O}_{\mathbb{G}_T}^{\mathsf{GGM}}$ is the final GGM oracle which takes as input anything outputted by $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$ and $\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}$, and outputs their bilinear pairing (i.e. the multiplication of the two discrete log polynomials).

Examining the Discrete Logs. It will be useful to us to examine the discrete logs of the outputs of the three GGM oracles. Depending on the value of $b \in$

$\{0,1\}$, the discrete log polynomial $p_1^{(*,b)}(\vec{\kappa})$ of $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}(Q_1^*)$ takes the form

$$\begin{aligned}
p_1^{(*,0)}(\vec{\kappa}) =& \alpha_1 + \sum_{i\in[5]}(\mu_{1,i}(\eta_1\rho_{1,i}\mathsf{sk}_{1,i}) + \tau_{1,i}(\eta_1\rho_{1,i}) + \mu_{2,i}(\eta_2\rho_{2,i}\mathsf{sk}_{2,i}) + \tau_{2,i}(\eta_2\rho_{2,i})) + \sum_{k\in[q_h]}\gamma_k(\eta_{2+k}) \\
&+ \sum_{k\in[q_\sigma]}\left(\beta_k\left(\eta^{(k)}\sum_{j\in[2]}\rho_j^{(k)}\mathsf{sk}^{(0)}_{\theta_k,3+j}\right) + \zeta_k\left(\eta^{(k)}\mathsf{sk}^{(0)}_{\theta_k,1} + \sum_{j\in[2]}p_{M_j^{(k)}}(\vec{\kappa})\mathsf{sk}^{(0)}_{\theta_k,1+j}\right)\right) \\
p_1^{(*,1)}(\vec{\kappa}) =& \alpha_1 + \sum_{i\in[5]}(\mu_{1,i}(\eta_1\rho_{1,i}\mathsf{sk}_{1,i}) + \tau_{1,i}(\eta_1\rho_{1,i}) + \mu_{2,i}(\eta_1\psi\rho_{1,i}\omega\mathsf{sk}_{1,i}) + \tau_{2,i}(\eta_1\psi\rho_{1,i})) + \sum_{k\in[q_h]}\gamma_k(\eta_{2+k}) \\
&+ \sum_{k\in[q_\sigma]}\left(\beta_k\left(\eta^{(k)}\sum_{j\in[2]}\rho_j^{(k)}\mathsf{sk}^{(1)}_{\theta_k,3+j}\right) + \zeta_k\left(\eta^{(k)}\mathsf{sk}^{(1)}_{\theta_k,1} + \sum_{j\in[2]}p_{M_j^{(k)}}(\vec{\kappa})\mathsf{sk}^{(1)}_{\theta_k,1+j}\right)\right)
\end{aligned}$$

where $\mathsf{sk}^{(0)}_{2,j} = \mathsf{sk}_{2,j}$, $\mathsf{sk}^{(b)}_{1,j} = \mathsf{sk}_{1,j}$, and $\mathsf{sk}^{(1)}_{2,j} = \omega\mathsf{sk}_{1,j}$. Similarly, depending on the hybrid, the discrete log polynomial $p_2^{(*,b)}(\vec{\kappa})$ of $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}(Q_2^*)$ takes the form

$$\begin{aligned}
p_2^{(*,0)}(\vec{\kappa}) &= \alpha_2 + \sum_{j\in[5]}(\nu_{1,j}(\mathsf{sk}_{1,j}) + \nu_{2,j}(\mathsf{sk}_{2,j})) \\
p_2^{(*,1)}(\vec{\kappa}) &= \alpha_2 + \sum_{j\in[5]}(\nu_{1,j}(\mathsf{sk}_{1,j}) + \nu_{2,j}(\omega\mathsf{sk}_{1,j}))
\end{aligned}$$

In Hybrid 1, the monomials 1, $\mathsf{sk}_{1,j}$, and $\mathsf{sk}_{2,j}$ are distinct, and in Hybrid 2, $p_2^{(*,1)}(\vec{\kappa}) = \alpha_2 + \sum_{j\in[5]}(\nu_{1,j} + \nu_{2,j}\,\omega)\mathsf{sk}_{1,j}$, where ω is an indeterminant. Hence, in either hybrid, $p_2^{(*,b)}(\vec{\kappa})$ is the zero polynomial if and only if all of the scalars α_2, $\nu_{1,j}$, and $\nu_{2,j}$ in Q_2^* are 0. In particular, $p_2^{(*,0)}(\vec{\kappa}) = 0$ if and only if $p_2^{(*,1)}(\vec{\kappa}) = 0$. We use this fact below. The combined query $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_T}(Q^*)$ has discrete log $p^{(*,b)}(\vec{\kappa}) = p_1^{(*,b)}(\vec{\kappa}) \cdot p_2^{(*,b)}(\vec{\kappa})$. In all these polynomials, we take $\vec{\kappa}$ to be the vector of indeterminants used in these polynomials, and say that each polynomial is an element of $\mathbb{Z}_p[\vec{\kappa}]$.

$$\vec{\kappa} = (\eta_1, \eta_2, \eta_{2+k}, \rho_{1,i}, \rho_{2,i}, \psi, \mathsf{sk}_{1,i}, \mathsf{sk}_{2,i}, \omega)_{k\in[q_h],i\in[5]},$$

where η_{2+k} is the discrete log of the kth output h_k of $\mathcal{O}^H$.

Proving the Claims. Now we can prove Claims 1 and 2.

Proof (of Claim 1). An adversary's view can differ between Hybrid 0 and Hybrid 1 if and only if some two queries q_1 and q_2 are equal in one hybrid but not equal in the other. Note that in Hybrid 1, the discrete log of every query is a polynomial $p(\vec{\kappa}) \in \mathbb{Z}_p[\vec{\kappa}]$. In Hybrid 0, each discrete log takes a value $p(\vec{a}) \in \mathbb{Z}_p$, where $\vec{a}$ denotes the values assigned to each indeterminant in $\vec{\kappa}$.

If $q_1(\vec{\kappa}) = q_2(\vec{\kappa})$ in Hybrid 1, then no matter what assignment is chosen, $q_1(\vec{a}) = q_2(\vec{a})$ will hold in Hybrid 0. On the other hand, if $q_1(\vec{\kappa}) \neq q_2(\vec{\kappa})$, whether $q_1(\vec{a}) = q_2(\vec{a})$ in Hybrid 0 depends on the assignment. We know that for $i \in [5]$,

$\rho_{1,i}$, $\rho_{2,i}$, $\mathsf{sk}_{1,i}$, and $\mathsf{sk}_{2,i}$ are all randomly sampled from $\mathbb{Z}_p^\times$ at the start of Hybrid 0, and if we model $\mathcal{O}^H$ as a random oracle, then η_1, η_2, and η_{2+k} for $k \in [q_h]$ are all also randomly sampled from $\mathbb{Z}_p^\times$. Finally, ψ and ω don't appear in either Hybrid 0 or Hybrid 1. Thus, we can apply the Schwartz-Zippel lemma, which tells us that if $q_1(\vec{\kappa}) - q_2(\vec{\kappa}) \neq 0$, there is a negligible chance that $q_1(\vec{a}) - q_2(\vec{a}) = 0$ for a random assignment $\vec{a}$, since all polynomials have degree bounded by a constant. A union bound over the polynomially many pairs of queries completes the argument. Therefore, with overwhelming probability, the adversary's view will be the same between Hybrid 0 and Hybrid 1. □

Proof (of Claim 2). The logic of this proof exactly mirrors that of the proof of Claim 1. Note that here, η_2, $\rho_{2,i}$, and $\mathsf{sk}_{2,i}$ don't appear in the output of any query. Instead, ψ and ω do, and these values are randomly sampled from $\mathbb{Z}_p^\times$. This allows us to apply the same reasoning as before. If $q_1(\vec{\kappa}) = q_2(\vec{\kappa})$ in Hybrid 2, then it must be true that $q_1(\vec{a}) = q_2(\vec{a})$ in Hybrid 3. Otherwise, if $q_1(\vec{\kappa}) \neq q_2(\vec{\kappa})$, then by the Schwartz-Zippel lemma, there is an overwhelming chance that $q_1(\vec{a}) \neq q_2(\vec{a})$. With overwhelming probability, the adversary's view is the same in both hybrids. □

Now what remains in proving Theorem 3 is to prove that Hybrid 1 and Hybrid 2 are computationally indistinguishable. For this, we give a reduction to the public key class-hiding of the plain version of $\mathbf{\Pi}_{\mathsf{MBGLS}}$.

Reduction. First, we know from [35], Lemma 31, that if an adversary is able to distinguish between two views, there exists an extractor that, given $\mathcal{A}$'s queries, can extract a distinguishing polynomial that is identically 0 in one hybrid and identically nonzero in the other. We call this extractor $\mathcal{E}$ and use it in our proof.

Suppose that there exists an adversary $\mathcal{A}$ that wins Game 1 with non-negligible advantage. We construct a reduction $\mathcal{B}$ that runs $\mathcal{A}$ as a subroutine to win Game 2, the game $\mathsf{PK\text{-}CLH}_{\mathcal{B}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ for the plain version of $\mathbf{\Pi}_{\mathsf{MBGLS}}$ (Definition 12 without the boxed elements). $\mathcal{B}$ functions as follows:

1. **Challenge Phase.** $\mathcal{B}$ challenges $\mathcal{A}$ to Game 1, sampling a uniform bit $b^\dagger \xleftarrow{\$} \{0,1\}$ and proceeding to act as the challenger in Hybrid $1 + b^\dagger$. As per the definition, $\mathcal{A}$ makes queries to each of $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$, $\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}$, $\mathcal{O}_{\mathbb{G}_T}^{\mathsf{GGM}}$, $\mathcal{O}^H$, and $\mathcal{O}^{\mathsf{Sign}}$. For each valid query, $\mathcal{B}$ responds honestly with encodings generated according to the definition of Hybrid $1 + b^\dagger$. $\mathcal{B}$ also keeps track of each valid query. On the other hand, $\mathcal{B}$ does *not* keep track of invalid queries (e.g. a $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$ query with the wrong number of scalar inputs, or a $\mathcal{O}^{\mathsf{Sign}}$ query that doesn't pass the $e(h^{\rho_j}, \hat{N}_j) = e(M_j, \hat{P})$ condition), as these queries have no output, and can be ignored WLOG. Once $\mathcal{A}$ has made all of its queries, it outputs its guess $b' \in \{0,1\}$ indicating that $\mathcal{A}$ believes it is playing Hybrid $1 + b'$.
2. **Extraction Phase.** $\mathcal{B}$ runs $\mathcal{E}$ on the set of $\mathcal{A}$'s successful queries, which returns a distinguishing query Q^* as a set of scalar inputs to $\mathcal{O}_{\mathbb{G}_T}^{\mathsf{GGM}}$. $\mathcal{B}$ computes $\mathcal{O}_{\mathbb{G}_T}^{\mathsf{GGM}}(Q^*)$ according to the definition in both Hybrid 1 and Hybrid 2, and examines the result's discrete log polynomial $p^*(\vec{\kappa})$ in each hybrid.

- If $p^*(\vec{\kappa})$ is the zero polynomial in one hybrid and a nonzero polynomial in the other, the extractor has done its job (this happens with non-negligible probability). $\mathcal{B}$ records the bit $c \in \{0,1\}$ such that $p^*(\vec{\kappa})$ is zero in Hybrid $1+c$, and continues with the next phase.
- Otherwise, $\mathcal{B}$ samples $b^* \xleftarrow{\$} \{0,1\}$, outputs b^* in Game 2, and terminates.

In the first case, the reduction continues with the next phase.

3. **Query Phase.** $\mathcal{B}$ plays Game 2 against the $\mathsf{PK\text{-}CLH}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}_{\mathcal{B}}$ challenger, who begins by secretly choosing $b \xleftarrow{\$} \{0,1\}$. The challenger then sends pk_1 and $\mathsf{pk}_2 = \mathsf{pk}_2^b$. $\mathcal{B}$'s goal is now to make all the same queries that $\mathcal{A}$ made. However, since $\vec{M}_1$, $\vec{T}_1$, $\vec{M}_2^{(b)}$, and $\vec{T}_2^{(b)}$ are given in Game 1 but not in Game 2, this may not be entirely straightforward. Thus, $\mathcal{B}$ loops through the valid queries made by $\mathcal{A}$ in order, and for each query does the following:
   - If it is a $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$ query, $\mathcal{B}$ attempts to compute it as a $\mathbb{G}_1$ element according to the expression given by $\mathcal{A}$'s inputs to the query. If successful, $\mathcal{B}$ stores the computed element with the encoding for use in future computations. Note that $\mathcal{B}$ will not be able to compute this query if it contains any $M_{1,i}$, $T_{1,i}$, $M_{2,i}^{(b)}$, or $T_{2,i}^{(b)}$ for some $i \in [5]$, or if it contains the output of some previous query that $\mathcal{B}$ did not successfully compute. In such a case, $\mathcal{B}$ simply moves on.
   - If it is a $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$ query, $\mathcal{B}$ computes it as a $\mathbb{G}_2$ element according to the expression given by $\mathcal{A}$'s inputs to the query, storing the result for use in future computations. This always succeeds, since every output of $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$ is a product of powers of $\hat{P}$ and the components of pk_1 and pk_2.
   - If it is a $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_T}$ query, $\mathcal{B}$ does nothing, since $\mathbb{G}_T$ elements are not the input to any other query, and the only $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_T}$ query that $\mathcal{B}$ needs is the distinguishing query Q^* returned by $\mathcal{E}$.
   - If it is a $\mathcal{O}^{\mathsf{Sign}}$ query, the input is a set of GGM encodings. $\mathcal{B}$ tries to find the previously-computed group element associated with each encoding. If successful, it sends these group elements as input to the signing oracle $\mathcal{O}_{\mathsf{Sign}_1}$ (if $\theta_k = 1$) or $\mathcal{O}_{\mathsf{Sign}_2^b}$ (if $\theta_k = 2$) of the $\mathsf{PK\text{-}CLH}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}_{\mathcal{B}}$ challenger, and stores the output for future use. However, if at least one of the GGM encodings corresponds to a query $\mathcal{B}$ did not successfully compute, then $\mathcal{B}$ fails, outputting $\perp$ and losing Game 2 (we'll show that this won't happen).
   - If it is a $\mathcal{O}^{H}$ query, the input is a string. If this string is a concatenation of two encodings for $\mathbb{G}_1$ elements followed by seven encodings for $\mathbb{G}_2$ elements (the form of the hashes computed by $\mathcal{O}^{\mathsf{Sign}}$), then $\mathcal{B}$ tries to find the computed group element associated with each encoding. If successful, $\mathcal{B}$ sends the string of those nine group elements as a $\mathcal{O}^{H}$ query to the $\mathsf{PK\text{-}CLH}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}_{\mathcal{B}}$ challenger, and stores the output for future use. If at least one of the GGM encodings corresponds to a query $\mathcal{B}$ did not successfully compute, or if the input does *not* have this form, $\mathcal{B}$ simply does not make the query.

   Once this is finished, $\mathcal{B}$ will have *tried* to replicate in Game 2 each query made by $\mathcal{A}$ in Game 1. (We'll show that the queries that were successful are enough to distinguish between the two hybrids.)

4. **Distinguishing Phase.** $\mathcal{B}$ partitions the scalars of Q^* into the scalars Q_1^* corresponding to $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$ and the scalars Q_2^* corresponding to $\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}$. It computes $D \in \mathbb{G}_1$ and $\hat{D} \in \mathbb{G}_2$ according to the expressions given by $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}(Q_1^*)$ and $\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}(Q_2^*)$, respectively, using the group elements stored in the Query Phase. If this fails, $\mathcal{B}$ outputs $\perp$ and loses Game 2 (we'll show that this won't happen). Otherwise, $\mathcal{B}$ computes $e(D, \hat{D}) \in \mathbb{G}_T$, outputs c if $e(D, \hat{D}) = 1_{\mathbb{G}_T}$, and outputs $1 - c$ otherwise. This ends the reduction to Game 2.

We make the following claims about this reduction:

Claim 3 *If $\mathcal{B}$ reaches the Query Phase, then the scalars $\tau_{\theta,i}$ and $\mu_{\theta,i}$ are 0 in Q^* for all $\theta \in \{1, 2\}$ and $i \in [5]$.*

Claim 4 *If $\mathcal{B}$ reaches the Query Phase, then $\mathcal{B}$ computes $e(D, \hat{D})$ in the Distinguishing Phase, and the discrete log of $e(D, \hat{D})$ is the value of $p^{(*,b)}(\vec{\kappa})$ at the values of Game 2, where b is the bit of the Game 2 challenger.*

Suppose that both claims hold, and that $\mathcal{B}$ reaches the Query Phase. If $b = c$, then $p^{(*,b)}(\vec{\kappa})$ is the zero polynomial, so $e(D, \hat{D}) = 1_{\mathbb{G}_T}$ and $\mathcal{B}$ outputs $c = b$. If $b \neq c$, then $p^{(*,b)}(\vec{\kappa})$ is a nonzero polynomial of bounded degree, and in Game 2 its indeterminants that occur are the secret keys, ω, and the outputs of the random oracle, which are uniformly random. By the Schwartz-Zippel lemma, $e(D, \hat{D}) \neq 1_{\mathbb{G}_T}$ with overwhelming probability, and $\mathcal{B}$ outputs $1 - c = b$. If $\mathcal{B}$ stops in the Extraction Phase, it guesses b with probability $\frac{1}{2}$. Since $\mathcal{E}$ succeeds with non-negligible probability if $\mathcal{A}$ has non-negligible advantage in Game 1, $\mathcal{B}$ has non-negligible advantage in Game 2, which contradicts the public key class-hiding of the plain version of $\mathbf{\Pi}_{\mathsf{MBGLS}}$ [40]. Therefore, proving both claims proves Theorem 3.

Proof of the Reduction. The proofs of these two claims require some groundwork. We first prove that the input $(\vec{M}^{(k)}, \vec{N}^{(k)}, \tau^{(k)})$ to the kth successful $\mathcal{O}^{\mathsf{Sign}}$ query made by $\mathcal{A}$ in Game 1 is independent of the outputs of other $\mathcal{O}^{\mathsf{Sign}}$ queries (Lemma 5), of any hashes except $h^{(k)}$ (Lemma 6), and of the tag secrets $\rho_{\theta,i}$ of the key encodings (Lemma 7), so that it can be computed by $\mathcal{B}$ in Game 2 (Lemma 8). Along the way, we prove that the output of a $\mathcal{O}^{\mathsf{Sign}}$ query is independent of the $\rho_{\theta,i}$ (Corollary 2). Finally, we prove that $\mathcal{O}_{\mathbb{G}_T}^{\mathsf{GGM}}(Q^*)$ is independent of any hashes except those output by a $\mathcal{O}^{\mathsf{Sign}}$ query (Lemma 9). All of this allows us to prove Claims 3 and 4.

Lemma 5. *If $(\vec{M}^{(k)}, \vec{N}^{(k)}, \tau^{(k)})$ is the input to a successful $\mathcal{O}^{\mathsf{Sign}}$ query in the GGM, then the expressions composing $\vec{M}^{(k)}$, $\vec{N}^{(k)}$, and $\tau^{(k)}$ are not functions of $h^{(i)}$, $b^{(i)}$, or $s^{(i)}$ for any $i \neq k$, where $\sigma^{(i)} = (h^{(i)}, b^{(i)}, s^{(i)})$ is the output of a different $\mathcal{O}^{\mathsf{Sign}}$ query.*

Proof. For this to be a successful $\mathcal{O}^{\mathsf{Sign}}$ query, the input $(\vec{M}^{(k)}, \vec{N}^{(k)}, \tau^{(k)} = (\rho_1^{(k)}, \rho_2^{(k)}))$ must be of the right form. We use this to show that each of $\vec{M}^{(k)}$, $\vec{N}^{(k)}$, and $\tau^{(k)}$ does not contain $h^{(i)}$, $b^{(i)}$, or $s^{(i)}$ for any $i \neq k$:

1. $\tau^{(k)} = (\rho_1^{(k)}, \rho_2^{(k)})$, where each $\rho_j^{(k)}$ is a scalar in $\mathbb{Z}_p^\times$. As a scalar chosen by $\mathcal{A}$, each $\rho_j^{(k)}$ is a constant in $\mathbb{Z}_p[\vec{\kappa}]$, so it contains none of the indeterminants of $h^{(i)}$, $b^{(i)}$, or $s^{(i)}$. Thus, $\tau^{(k)}$ is not a function of any previously-queried signature.
2. $\vec{N}^{(k)} = (\hat{N}_1^{(k)}, \hat{N}_2^{(k)})$, where each $\hat{N}_j^{(k)} \in \mathbb{G}_2$ is the result of a query to $\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}$. By definition of $\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}$, it is not possible for any output of this oracle to output an expression containing $h^{(i)}$, $b^{(i)}$, or $s^{(i)}$, since these are not elements of the second source group. Thus, $\vec{N}^{(k)}$ is not a function of any previously-queried signature.
3. $\vec{M}^{(k)} = (M_1^{(k)}, M_2^{(k)})$, where each $M_j^{(k)} \in \mathbb{G}_1$ is the result of a query to $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$. Here, we use the fact that a well-formed $\mathcal{O}^{\mathsf{Sign}}$ input must satisfy the following equation, where $h^{(k)} = H(P^{\rho_1^{(k)}} \| P^{\rho_2^{(k)}} \| \hat{N}_1^{(k)} \| \hat{N}_2^{(k)} \| \mathsf{pk}_{\theta_k})$:

$$e((h^{(k)})^{\rho_j^{(k)}}, \hat{N}_j^{(k)}) = e(M_j^{(k)}, \hat{P}) \tag{7}$$

   We can examine the discrete log of this equation:

$$\eta^{(k)} \rho_j^{(k)} p_{\hat{N}_j^{(k)}}(\vec{\kappa}) = p_{M_j^{(k)}}(\vec{\kappa}) \tag{8}$$

   where $\eta^{(k)}$, $p_{\hat{N}_j^{(k)}}(\vec{\kappa})$, and $p_{M_j^{(k)}}(\vec{\kappa})$ are the discrete logs of $h^{(k)}$, $\hat{N}_j^{(k)}$, and $M_j^{(k)}$, respectively. $\eta^{(k)}$ is its own indeterminant, distinct from the $\eta^{(i)}$ for $i \neq k$, and we've just shown that $\rho_j^{(k)}$ and $p_{\hat{N}_j^{(k)}}(\vec{\kappa})$ are not functions of $h^{(i)}$, $b^{(i)}$, or $s^{(i)}$. Thus, the LHS of the equation doesn't contain any of these terms. For the equation to be satisfied, the RHS—and by extension $p_{M_j^{(k)}}(\vec{\kappa})$—must also not contain $h^{(i)}$, $b^{(i)}$, or $s^{(i)}$. Therefore, $\vec{M}^{(k)}$ is not a function of any previously-queried signature.

We've shown that $M^{(k)}$, $N^{(k)}$, and $\tau^{(k)}$ all do not contain $h^{(i)}$, $b^{(i)}$, or $s^{(i)}$ for any $i \neq k$, which concludes our proof of Lemma 5. □

Lemma 6. *If $(\vec{M}^{(k)}, \vec{N}^{(k)}, \tau^{(k)})$ is the input to a successful $\mathcal{O}^{\mathsf{Sign}}$ query in the GGM, then the expressions composing $\vec{M}^{(k)}$, $\vec{N}^{(k)}$, and $\tau^{(k)}$ are not functions of any hash h_i except $h_i = h^{(k)}$, where $h^{(k)}$ is part of the output $\sigma^{(k)} = (h^{(k)}, b^{(k)}, s^{(k)})$ of this $\mathcal{O}^{\mathsf{Sign}}$ query.*

Proof. For a $\mathcal{O}^{\mathsf{Sign}}$ query to return a signature (and not $\perp$), its input $(\vec{M}^{(k)}, \vec{N}^{(k)}, \tau^{(k)} = (\rho_1^{(k)}, \rho_2^{(k)}))$ must be a valid message. That is, for each $j \in [2]$, $(\vec{M}^{(k)}, \vec{N}^{(k)}, \tau^{(k)})$ must satisfy the verification equation

$$e((h^{(k)})^{\rho_j^{(k)}}, \hat{N}_j^{(k)}) = e(M_j^{(k)}, \hat{P}) \tag{9}$$

whose discrete log can be written as

$$\eta^{(k)} \rho_j^{(k)} p_{\hat{N}_j^{(k)}}(\vec{\kappa}) = p_{M_j^{(k)}}(\vec{\kappa}) \tag{10}$$

where $\eta^{(k)}$, $p_{\hat{N}_j^{(k)}}(\vec{\kappa})$, and $p_{M_j^{(k)}}(\vec{\kappa})$ are the discrete logs of $h^{(k)}$, $\hat{N}_j^{(k)}$, and $M_j^{(k)}$, respectively. Note that $M_j^{(k)}$ and $\hat{N}_j^{(k)}$ are the results of $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$ and $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$ queries, hence our representation of their discrete logs as polynomials in $\mathbb{Z}_p[\vec{\kappa}]$.

Note that $\rho_j^{(k)}$ is a scalar in $\mathbb{Z}_p^\times$ and $\hat{N}_j^{(k)}$ is a group element in $\mathbb{G}_2$, so neither can be a function of some hash $h_i \in \mathbb{G}_1$. So suppose then that $M_j^{(k)}$ is a function of h_i. We can write $p_{M_j^{(k)}}(\vec{\kappa}) = \gamma_i \eta_i + \cdots$, where γ_i is a scalar and η_i is the discrete log of h_i. Using this, we rewrite the above equation as

$$\eta^{(k)} \rho_j^{(k)} p_{\hat{N}_j^{(k)}}(\vec{\kappa}) = \gamma_i \eta_i + \cdots \tag{11}$$

Since η_i is present on the RHS of the equation, it must also be present on the LHS if we wish to satisfy the condition. However, as we've already stated, $\rho_j^{(k)}$ is a scalar, so it cannot contain the indeterminant η_i. Additionally, $\hat{N}_j^{(k)}$ is the output of a $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$ query, so by definition $p_{\hat{N}_j^{(k)}}(\vec{\kappa})$ can't contain the indeterminant η_i.

Therefore, for the equation above to be satisfied, it must be the case that $\eta^{(k)} = \eta_i$. We conclude that $\vec{M}^{(k)}$, $\vec{N}^{(k)}$, and $\tau^{(k)}$ are not functions of any hash other than $h^{(k)}$, which concludes our proof of Lemma 6. □

Lemma 7. *If $(\vec{M}^{(k)}, \vec{N}^{(k)}, \tau^{(k)})$ is the input to a successful $\mathcal{O}^{\mathsf{Sign}}$ query made by $\mathcal{A}$, then the expressions composing $\vec{M}^{(k)}$, $\vec{N}^{(k)}$, and $\tau^{(k)}$ are not functions of the tag secrets $\rho_{\theta,i}$ of the key encodings.*

Proof. For this to be a successful $\mathcal{O}^{\mathsf{Sign}}$ query, the input $(\vec{M}^{(k)}, \vec{N}^{(k)}, \tau^{(k)} = (\rho_1^{(k)}, \rho_2^{(k)}))$ must be of the right form. We use this to show that none of $\vec{M}^{(k)}$, $\vec{N}^{(k)}$, and $\tau^{(k)}$ contains any $\rho_{\theta,i}$:

1. $\tau^{(k)} = (\rho_1^{(k)}, \rho_2^{(k)})$, where each $\rho_j^{(k)}$ is a scalar in $\mathbb{Z}_p^\times$. Since the $\rho_{\theta,i}$ are indeterminants whose values are sampled only at the very end of Game 1, it is not possible for $\rho_j^{(k)}$ to be a function of them.
2. $\vec{N}^{(k)} = (\hat{N}_1^{(k)}, \hat{N}_2^{(k)})$, where each $\hat{N}_j^{(k)} \in \mathbb{G}_2$ is the result of a query to $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$. By definition of $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$ (i.e. it only outputs encodings of polynomials representations of queries in the second source group) it is not possible for any output of this oracle to contain an indeterminant $\rho_{\theta,i}$, since these do not appear in the polynomial of any element of the second source group.
3. $\vec{M}^{(k)} = (M_1^{(k)}, M_2^{(k)})$, where each $M_j^{(k)} \in \mathbb{G}_1$ is the result of a query to $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$. Here, we use the fact that a well-formed $\mathcal{O}^{\mathsf{Sign}}$ input satisfies (7), whose discrete log is

$$\eta^{(k)} \rho_j^{(k)} p_{\hat{N}_j^{(k)}}(\vec{\kappa}) = p_{M_j^{(k)}}(\vec{\kappa}) \tag{12}$$

where $\eta^{(k)}$, $p_{\hat{N}_j^{(k)}}(\vec{\kappa})$, and $p_{M_j^{(k)}}(\vec{\kappa})$ are the discrete logs of $h^{(k)}$, $\hat{N}_j^{(k)}$, and $M_j^{(k)}$, respectively. $\eta^{(k)}$ is a hash indeterminant, distinct from the discrete

logs η_θ of the hashes in the key encodings, and we've just shown that $\rho_j^{(k)}$ and $p_{\hat{N}_j^{(k)}}(\vec{\kappa})$ are not functions of any $\rho_{\theta,i}$. Thus, the LHS of the equation doesn't contain any of these indeterminants. For the equation to be satisfied, the RHS, $p_{M_j^{(k)}}(\vec{\kappa})$, must not contain them either.

We've shown that none of $\vec{M}^{(k)}$, $\vec{N}^{(k)}$, and $\tau^{(k)}$ contains any $\rho_{\theta,i}$, which concludes our proof of Lemma 7. □

Corollary 2. *If $(\vec{M}^{(k)}, \vec{N}^{(k)}, \tau^{(k)})$ is the input to a successful $\mathcal{O}^{\mathsf{Sign}}$ query made by $\mathcal{A}$, then the discrete logs of the output $\sigma^{(k)} = (h^{(k)}, b^{(k)}, s^{(k)})$ are not functions of the tag secrets $\rho_{\theta,i}$ of the key encodings.*

Proof. This follows directly from Lemma 7 and the definition of $\mathcal{O}^{\mathsf{Sign}}$. As a reminder, on input $(M^{(k)}, N^{(k)}, \tau^{(k)} = (\rho_1^{(k)}, \rho_2^{(k)}))$, $\mathcal{O}^{\mathsf{Sign}}$ outputs $\sigma^{(k)} = (h^{(k)}, b^{(k)}, s^{(k)})$, where

$$
\begin{aligned}
h^{(k)} &= H(P^{\rho_1^{(k)}} \| P^{\rho_2^{(k)}} \| \hat{N}_1^{(k)} \| \hat{N}_2^{(k)} \| \mathsf{pk}_{\theta_k}) \\
b^{(k)} &= \prod_{j \in [2]} (h^{(k)})^{\rho_j^{(k)} \mathsf{sk}_{\theta_k,3+j}^{(b)}} \\
s^{(k)} &= (h^{(k)})^{\mathsf{sk}_{\theta_k,1}^{(b)}} \prod_{j \in [2]} (M_j^{(k)})^{\mathsf{sk}_{\theta_k,1+j}^{(b)}}
\end{aligned}
$$

for $b \in \{0,1\}$ (depending on which hybrid $\mathcal{A}$ is in).

$h^{(k)}$ is the output of a random oracle whose discrete log is the indeterminant $\eta^{(k)}$, so it is not a function of any $\rho_{\theta,i}$.

$b^{(k)}$ and $s^{(k)}$ are composed of $h^{(k)}$, which we just showed is independent of the $\rho_{\theta,i}$. They are also composed of $\rho_j^{(k)}$ and $M_j^{(k)}$ for $j \in [2]$, but Lemma 7 tells us that these are not functions of any $\rho_{\theta,i}$ either.

Finally, the only other terms that compose $b^{(k)}$ and $s^{(k)}$ are $\mathsf{sk}_{\theta_k,j}^{(b)}$ for $j \in [5]$, where $\mathsf{sk}_{1,j}^{(b)} = \mathsf{sk}_{1,j}$, $\mathsf{sk}_{2,j}^{(0)} = \mathsf{sk}_{2,j}$, and $\mathsf{sk}_{2,j}^{(1)} = \omega \mathsf{sk}_{1,j}$. In all cases, $\mathsf{sk}_{1,j}$, $\mathsf{sk}_{2,j}$, and ω are indeterminants distinct from the $\rho_{\theta,i}$.

Thus, no discrete log in $\sigma^{(k)}$ contains any $\rho_{\theta,i}$. This concludes our proof of Corollary 2. □

Lemma 8. *If $(\vec{M}^{(k)}, \vec{N}^{(k)}, \tau^{(k)})$ is the input to a successful $\mathcal{O}^{\mathsf{Sign}}$ query made by $\mathcal{A}$, and $\mathcal{B}$ has replicated all earlier queries as described in the Query Phase, then $\mathcal{B}$ can compute $\vec{M}^{(k)}$, $\vec{N}^{(k)}$, and $\tau^{(k)}$ in Game 2.*

Proof. The fact that the query $\mathcal{O}^{\mathsf{Sign}}(\vec{M}^{(k)}, \vec{N}^{(k)}, \tau^{(k)})$ was successful tells us that the inputs are of the correct form. Using this, we show that each term can be computed:

1. $\tau^{(k)} = (\rho_1^{(k)}, \rho_2^{(k)})$, where each $\rho_j^{(k)}$ is a scalar in $\mathbb{Z}_p^\times$. Since $\mathcal{B}$ has access to $\mathcal{A}$'s query, it knows these scalars.

2. $\vec{N}^{(k)} = (\hat{N}_1^{(k)}, \hat{N}_2^{(k)})$, where each $\hat{N}_j^{(k)} \in \mathbb{G}_2$ is the result of a query to $\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}$. As a reminder, $\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}$ outputs expressions of the following form:

$$\hat{P}^{\alpha} \prod_{i \in [5]} (\mathsf{pk}_{1,i})^{\nu_{1,i}} (\mathsf{pk}_{2,i}^{(b)})^{\nu_{2,i}}$$

$\mathcal{B}$ is given the public keys at the start of Game 2, so it will always be able to compute $\vec{N}^{(k)}$ as a vector of elements in $\mathbb{G}_2$.

3. $\vec{M}^{(k)} = (M_1^{(k)}, M_2^{(k)})$, where each $M_j^{(k)} \in \mathbb{G}_1$ is the result of a query to $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$. As a reminder, $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$ outputs expressions of the following form:

$$P^{\alpha} \prod_{i \in [5]} (M_{1,i})^{\mu_{1,i}} (T_{1,i})^{\tau_{1,i}} (M_{2,i}^{(b)})^{\mu_{2,i}} (T_{2,i}^{(b)})^{\tau_{2,i}} \prod_{i \in [q_h]} h_i^{\gamma_i} \prod_{k \in [q_\sigma]} (b^{(k)})^{\beta_k} (s^{(k)})^{\zeta_k}$$

There are three reasons that could cause $\mathcal{B}$ to fail in the computation of a $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$ expression. Each reason is listed and eliminated below:

- If the expression contains a key encoding element $T_{\theta,i}$ or $M_{\theta,i}$ with a nonzero scalar, $\mathcal{B}$ can't compute it as an element of $\mathbb{G}_1$, since these elements are not given to it in Game 2. However, Lemma 7 tells us that the discrete logs of $\vec{M}^{(k)}$ are not functions of any $\rho_{\theta,i}$. The discrete logs of the $T_{\theta,i}$ and $M_{\theta,i}$ are distinct monomials that contain some $\rho_{\theta',i}$, while those of hashes and of earlier signatures contain none (Corollary 2). Hence they cannot cancel, and all of their scalars in the expressions for $\vec{M}^{(k)}$ are 0.
- If the expression contains a signature component $h^{(i)}$, $b^{(i)}$, or $s^{(i)}$ from a previous $\mathcal{O}^{\mathsf{Sign}}$ query that $\mathcal{B}$ failed to replicate in Game 2, then $\mathcal{B}$ will fail to compute this expression. However, Lemma 5 tells us that the expressions composing $M^{(k)}$ are not functions of $h^{(i)}$, $b^{(i)}$, or $s^{(i)}$ for any $i \neq k$, so this is not the case.
- Finally, if the expression contains a hash h_i from a previous $\mathcal{O}^H$ query that $\mathcal{B}$ failed to replicate in Game 2, then $\mathcal{B}$ will fail to compute this expression. However, Lemma 6 tells us that the expressions composing $M^{(k)}$ are not functions of any hash h_i except $h_i = h^{(k)} = H(P^{\rho_1^{(k)}} \| P^{\rho_2^{(k)}} \| \hat{N}_1^{(k)} \| \hat{N}_2^{(k)} \| \mathsf{pk}_{\theta_k})$. We've just shown that each $\rho_j^{(k)}$ and $\hat{N}_j^{(k)}$ for $j \in [2]$ can be computed by $\mathcal{B}$, and $\mathcal{B}$ knows pk_{θ_k}, so it can query $h^{(k)}$ to the random oracle of Game 2.

Thus, it is guaranteed that $\mathcal{B}$ will always be able to compute $\vec{M}^{(k)}$ as a vector of elements in $\mathbb{G}_1$.

We've shown that each of $\vec{M}^{(k)}$, $\vec{N}^{(k)}$, and $\tau^{(k)}$ can be computed by $\mathcal{B}$ in Game 2, which concludes our proof of Lemma 8. □

Lemma 9. *If $\mathcal{B}$ reaches the Query Phase, then $\mathcal{O}_{\mathbb{G}_T}^{\mathsf{GGM}}(Q^*)$ is not a function of any hash h_i not of the form $h_i = h^{(k)}$ for some $k \in [q_\sigma]$, where $h^{(k)}$ is part of the output $\sigma^{(k)} = (h^{(k)}, b^{(k)}, s^{(k)})$ of some $\mathcal{O}^{\mathsf{Sign}}$ query. That is, the scalar γ_i of every other hash h_i is 0 in Q^*.*

Proof. Let Hybrid $1 + c$ be the hybrid in which $p^{(*,c)}(\vec{\kappa}) = p_1^{(*,c)}(\vec{\kappa}) \cdot p_2^{(*,c)}(\vec{\kappa})$ is the zero polynomial, and let U be the set of indices i of hashes h_i that are not of the form $h^{(k)}$. We split $p_1^{(*,c)}(\vec{\kappa}) = g(\vec{\kappa}) + r(\vec{\kappa})$, where $g(\vec{\kappa}) = \sum_{i \in U} \gamma_i \eta_{2+i}$ collects the terms of these hashes and $r(\vec{\kappa})$ collects all other terms. No term of $r(\vec{\kappa})$ contains an indeterminant η_{2+i} with $i \in U$; the key encodings contain only η_1 and η_2, which $\mathcal{A}$ obtains from $\mathcal{O}^H$ only with negligible probability. The other hashes are distinct indeterminants, and the outputs of $\mathcal{O}^{\mathsf{Sign}}$ contain no hash other than $h^{(k)}$ by Lemma 6. No term of $p_2^{(*,c)}(\vec{\kappa})$ contains a hash indeterminant at all.

Hence, every term of $g(\vec{\kappa})p_2^{(*,c)}(\vec{\kappa})$ contains some η_{2+i} with $i \in U$, and no term of $r(\vec{\kappa})p_2^{(*,c)}(\vec{\kappa})$ does. Thus the two products cannot cancel, and $g(\vec{\kappa})p_2^{(*,c)}(\vec{\kappa}) = 0$. Since $\mathbb{Z}_p[\vec{\kappa}]$ has no zero divisors, $g(\vec{\kappa}) = 0$ or $p_2^{(*,c)}(\vec{\kappa}) = 0$. If $p_2^{(*,c)}(\vec{\kappa}) = 0$, then also $p_2^{(*,1-c)}(\vec{\kappa}) = 0$ ("Examining the Discrete Logs"), so $p^{(*,1-c)}(\vec{\kappa}) = 0$, contradicting the fact that $p^*(\vec{\kappa})$ is nonzero in the other hybrid. Thus $g(\vec{\kappa}) = 0$, and since the η_{2+i} are distinct indeterminants, $\gamma_i = 0$ for all $i \in U$. This concludes our proof of Lemma 9. □

Proof (of Claim 3). Since $\mathcal{B}$ reached the Query Phase, $p^*(\vec{\kappa})$ is the zero polynomial in Hybrid $1 + c$ and a nonzero polynomial in the other hybrid. As in the proof of Lemma 9, we split $p_1^{(*,b)}(\vec{\kappa})$ into the terms that come from the key encodings and the rest:

$$\begin{aligned} p_1^{(*,b)}(\vec{\kappa}) &= a^{(b)}(\vec{\kappa}) + r^{(b)}(\vec{\kappa}), \\ a^{(b)}(\vec{\kappa}) &= \sum_{\theta \in \{1,2\}} \sum_{i \in [5]} \left(\tau_{\theta,i}\, p_{T_{\theta,i}}(\vec{\kappa}) + \mu_{\theta,i}\, p_{M_{\theta,i}}(\vec{\kappa}) \right), \end{aligned}$$

where $p_{T_{\theta,i}}(\vec{\kappa})$ and $p_{M_{\theta,i}}(\vec{\kappa})$ are the discrete logs of $T_{\theta,i}$ and $M_{\theta,i}$ in the respective hybrid, and $r^{(b)}(\vec{\kappa})$ collects the terms with the scalars α_1, γ_k, β_k, and ζ_k. Then $p^{(*,b)}(\vec{\kappa}) = a^{(b)}(\vec{\kappa})p_2^{(*,b)}(\vec{\kappa}) + r^{(b)}(\vec{\kappa})p_2^{(*,b)}(\vec{\kappa})$. We make two observations.

1. Every term of $a^{(b)}(\vec{\kappa})$ contains one of the indeterminants $\rho_{1,i}$ or $\rho_{2,i}$, while no term of $r^{(b)}(\vec{\kappa})$ or $p_2^{(*,b)}(\vec{\kappa})$ does. This is because $r^{(b)}(\vec{\kappa})$ is composed of α_1, hashes, and signature outputs, which are not functions of the $\rho_{\theta,i}$ by Corollary 2, and no element of $\mathbb{G}_2$ contains these indeterminants. Thus, the two products cannot cancel each other. In particular, if $p^{(*,b)}(\vec{\kappa}) = 0$, then $a^{(b)}(\vec{\kappa})p_2^{(*,b)}(\vec{\kappa}) = 0$.
2. In each hybrid, $a^{(b)}(\vec{\kappa}) = 0$ if and only if all of the scalars $\tau_{\theta,i}$ and $\mu_{\theta,i}$ are 0, since the discrete logs of the key encodings are distinct monomials: $\eta_1\rho_{1,i}$, $\eta_1\rho_{1,i}\mathsf{sk}_{1,i}$, $\eta_2\rho_{2,i}$, and $\eta_2\rho_{2,i}\mathsf{sk}_{2,i}$ in Hybrid 1, and $\eta_1\rho_{1,i}$, $\eta_1\rho_{1,i}\mathsf{sk}_{1,i}$, $\eta_1\psi\rho_{1,i}$, and $\eta_1\psi\rho_{1,i}\omega\mathsf{sk}_{1,i}$ in Hybrid 2.

By the first observation, $a^{(c)}(\vec{\kappa})p_2^{(*,c)}(\vec{\kappa}) = 0$. Since $\mathbb{Z}_p[\vec{\kappa}]$ has no zero divisors, either $a^{(c)}(\vec{\kappa}) = 0$ or $p_2^{(*,c)}(\vec{\kappa}) = 0$. If $p_2^{(*,c)}(\vec{\kappa}) = 0$, then $p_2^{(*,1-c)}(\vec{\kappa}) = 0$ as well ("Examining the Discrete Logs"), so $p^{(*,1-c)}(\vec{\kappa}) = p_1^{(*,1-c)}(\vec{\kappa}) \cdot p_2^{(*,1-c)}(\vec{\kappa}) = 0$, contradicting the fact that $p^*(\vec{\kappa})$ is nonzero in the other hybrid. Thus $a^{(c)}(\vec{\kappa}) = 0$,

and by the second observation, all of the scalars $\tau_{\theta,i}$ and $\mu_{\theta,i}$ are 0 in Q^*. This concludes our proof of Claim 3. □

Proof (of Claim 4). For $\mathcal{B}$ to have reached the Query Phase, $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_T}(Q^*)$ must be a valid GGM query, so $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}(Q_1^*)$ and $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}(Q_2^*)$ are valid GGM queries. We show that $\mathcal{B}$ can compute them as group elements $D \in \mathbb{G}_1$ and $\hat{D} \in \mathbb{G}_2$.

$\hat{D} \in \mathbb{G}_2$ is computed according to the expression given by $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}(Q_2^*)$. As in the proof of Lemma 8, the expressions output by $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$ only involve $\hat{P}$ and the public keys, so $\mathcal{B}$ can compute $\hat{D}$.

$D \in \mathbb{G}_1$, on the other hand, is computed according to the expression given by $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}(Q_1^*)$. By inspecting the definition of $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$ we see three reasons that could cause $\mathcal{B}$ to fail in the computation of a $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$ expression. Each reason is listed and eliminated below:

1. If the expression contains at least one of $T_{1,i}$, $M_{1,i}$, $T_{2,i}^{(b)}$, or $M_{2,i}^{(b)}$ with a nonzero scalar, $\mathcal{B}$ can't compute it as an element of $\mathbb{G}_1$, since these terms are not given by the challenger in Game 2. However, by Claim 3, the scalars $\tau_{\theta,i}$ and $\mu_{\theta,i}$ are 0 in Q^*.
2. If the expression contains a signature component $h^{(k)}$, $b^{(k)}$, or $s^{(k)}$ from a $\mathcal{O}^{\mathsf{Sign}}$ query that $\mathcal{B}$ failed to replicate in Game 2, then $\mathcal{B}$ will fail to compute this expression. For $\mathcal{B}$ to fail when making a $\mathcal{O}^{\mathsf{Sign}}$ query, it must have failed to compute one of the expressions in the input $(\vec{M}^{(k)}, \vec{N}^{(k)}, \tau^{(k)})$. However, by induction over the queries, Lemma 8 tells us that $\mathcal{B}$ can compute the input to every $\mathcal{O}^{\mathsf{Sign}}$ query made by $\mathcal{A}$, so this is not the case.
3. Finally, if the expression contains a hash h_i from a $\mathcal{O}^{H}$ query that $\mathcal{B}$ failed to replicate in Game 2, then $\mathcal{B}$ will fail to compute this expression. However, Lemma 9 tells us that any hash h_i that appears in Q^* with a nonzero scalar has the form $h_i = h^{(k)}$, which $\mathcal{B}$ obtains in Game 2 as part of the output of the replicated $\mathcal{O}^{\mathsf{Sign}}$ query.

So $\mathcal{B}$ computes $e(D, \hat{D}) \in \mathbb{G}_T$ in Game 2. Moreover, D and $\hat{D}$ are computed from the same expressions as in Game 1, and the elements they involve (the public keys, the hashes $h^{(k)}$, and the signatures returned by the challenger) have as discrete logs the polynomials of Hybrid $1+b$, evaluated at the values of Game 2. Hence, the discrete log of $e(D, \hat{D})$ is $p^{(*,b)}(\vec{\kappa})$ evaluated at these values. This concludes our proof of Claim 4. □

Conclusion of Theorem 3. We have proven Claims 3 and 4. As shown after their statements, $\mathcal{B}$ then has non-negligible advantage in Game 2 whenever $\mathcal{A}$ has non-negligible advantage in Game 1, contradicting the public key class-hiding of the plain version of $\mathbf{\Pi}_{\mathsf{MBGLS}}$. Together with Claims 1 and 2, this proves the public key class-hiding of the cross-scheme correct version of $\mathbf{\Pi}_{\mathsf{MBGLS}}$ in the GGM and ROM. □

E Threshold Delegatable Anonymous Credentials

In this section, we formally define threshold delegatable anonymous credentials (TDAC) (Appendix E.1), present our construction (Appendix E.2), and prove its security (Appendix E.3); Section 4 gives an overview. Although we do not formalize revocation, our construction is compatible with the generic transformation of [35].

E.1 Syntax and Security Definitions

Definition 13 (Syntax of TDAC). *A threshold delegatable anonymous credentials scheme* TDAC *consists of the* PPT *algorithms and interactive protocols* (Setup, IssKeyGen, UserKeyGen, Issue, Receive, CredComb, UIssue, UReceive, UCredComb, Prove, Verify)*, as follows.*

- $\mathsf{Setup}(1^\lambda, 1^L, 1^n, 1^t) \to \mathsf{pp}$*, the setup algorithm, takes the security parameter* λ*, the length* L *of credential chains (from the root down to the user), the number* n *of parties that share each issuer key, and the threshold* t*, and outputs the public parameters* pp*.*
- $\mathsf{IssKeyGen}(\mathsf{pp}, L', \mathsf{pk}') \to (\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]})$*, the issuer key generation algorithm, takes* pp*, a level* $L' \in [L-1]$*, and the public key* pk' *of the issuer at level* $L'-1$ *that will certify the new key (*$\mathsf{pk}' = \perp$ *for the root,* $L' = 1$*). It outputs the issuer's public key* pk *and one secret share* sk_i *for each of the* n *parties.*
- $\mathsf{UserKeyGen}(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{sk})$*, the user key generation algorithm, outputs a user's key pair; users are not thresholdized.*
- $\langle \mathsf{Issue}(\mathsf{sk}_{\mathsf{I},i}, \mathsf{cred}_\mathsf{I}, L') \leftrightarrow \mathsf{Receive}(\mathsf{sk}_{\mathsf{R},j}, L')\rangle \to \mathsf{pcred}_{\mathsf{R},i}$*, the issuance protocol, is run between party* i *of an issuer* I*, on input its share* $\mathsf{sk}_{\mathsf{I},i}$ *and the issuer's credential* cred_I*, and party* j *of a new issuer* R *at level* L'*, on input its share* $\mathsf{sk}_{\mathsf{R},j}$*. On success, the receiver obtains a partial credential* $\mathsf{pcred}_{\mathsf{R},i}$*.*
- $\mathsf{CredComb}(\{\mathsf{pcred}_{\mathsf{R},i}\}_{i\in\mathfrak{T}}, \mathsf{pk}_\mathsf{R}) \to \mathsf{cred}_\mathsf{R}$*, the credential combination algorithm, combines partial credentials from a set* $\mathfrak{T}$ *of* t *parties of* I *into the credential* cred_R *of* R*, or outputs* $\perp$*.*
- $\langle \mathsf{UIssue}(\mathsf{sk}_{\mathsf{I},i}, \mathsf{cred}_\mathsf{I}) \leftrightarrow \mathsf{UReceive}(\mathsf{sk})\rangle \to \mathsf{pcred}_i$*, the user issuance protocol, is run between party* i *of an issuer* I *at level* $L-1$ *and a user with secret key* sk*, who obtains a partial credential* pcred_i*.*
- $\mathsf{UCredComb}(\{\mathsf{pcred}_i\}_{i\in\mathfrak{T}}, \mathsf{pk}) \to \mathsf{cred}$*, the user credential combination algorithm, combines partial credentials from* t *parties of* I *into the user's credential* cred*, or outputs* $\perp$*.*
- $\langle \mathsf{Prove}(\mathsf{sk}, \mathsf{cred}) \leftrightarrow \mathsf{Verify}(\mathsf{pk})\rangle \to \{0,1\}$*, the showing protocol, is run between a user with secret key* sk *and credential* cred *and a verifier with the root public key* pk*. The verifier outputs* 1 *if it accepts that the user holds a credential whose chain starts at* pk*.*

A TDAC *scheme must satisfy correctness (Definition 14), anonymity (Definition 15), and unforgeability (Definition 16).*

Oracles. Before presenting our security definitions and corresponding games, we define the oracles through which the adversary interacts with the challenger and the honest parties. The formal descriptions appear in Figure 7; we provide an informal overview below.

- $\mathcal{O}^{\mathsf{CreateHI}}$: The adversary requests the creation of a new honest issuer and the delegation of issuing authority to it. It specifies the handle id_I of an existing honest issuer (*e.g.*, the root) together with t of its parties $\mathfrak{T}_\mathsf{I}$, a fresh handle id_R for the new issuer, a receiving party j, and a set $\mathfrak{T}_{\mathsf{Adv}}$ of $t-1$ parties of the new issuer to corrupt. The challenger generates the new issuer's keys for the level below id_I, gives the adversary the new public key and the key shares of the parties in $\mathfrak{T}_{\mathsf{Adv}}$, and runs the issuance protocol between the parties $\mathfrak{T}_\mathsf{I}$ and party j. If the receiving handle has already been used, the challenger aborts.
- $\mathcal{O}^{\mathsf{CreateHU}}$: The same for a new honest user, who is not thresholdized; the adversary learns only the user's public key.
- $\mathcal{O}^{\mathsf{IssueToAdv}}$ and $\mathcal{O}^{\mathsf{UIssueToAdv}}$: In the anonymity game, these oracles allow the adversary to obtain a credential, as an issuer or as a user, from t parties of an honest issuer.
- $\mathcal{O}^{\mathsf{IssueFromAdv}}$ and $\mathcal{O}^{\mathsf{UIssueFromAdv}}$: These oracles allow the adversary, acting as an issuer with public key pk', to issue a credential to a new honest issuer or user id_R. The challenger generates and records the keys of id_R (for an issuer, with pk' as the certifying key) and runs the receiver's side of the issuance protocol.[3]
- $\mathcal{O}^{\mathsf{ProveToAdv}}$: Here, the adversary acts as verifier in an interaction with an honest user id_P, who proves possession of a valid credential.

Throughout the game, the challenger allows the adversary to specify "handles" to reference honest users and issuers in the game. These can be thought of as integers and are labeled with id. We use subscripts to distinguish these (*e.g.*, id_I for an issuer handle). To properly simulate multiple honest users, the challenger keeps track of the following maps $(\mathsf{SK}, \mathsf{CRED}, \mathsf{PK})$ that map handles of honest issuers and users to their keys and credentials:

- SK: Stores secret keys. For issuer handles, SK maps each handle to a set of threshold secret-key shares; for user handles, it maps to a single secret key.
- CRED: Maps handles to their associated credentials for both issuers and users. For a handle id, the length $|\mathsf{CRED}[\mathsf{id}]|$ determines the delegation level in $[L]$ of the corresponding party.
- PK: Maps the handles of honest issuers and users to canonical representations of their public keys. To populate the PK map, we use an unbounded extractor $\mathcal{E}_{\mathcal{PK}}$ to derive the canonical representation of a public key. By "canonical representation" we mean that the extractor will extract the same value to

[3] In contrast to [35], we do not model issuing forgeries (i.e., cases where a malicious issuer invokes $\mathcal{O}^{\mathsf{IssueFromAdv}}$ using a credential it was not legitimately issued). This is because a credential issued using a forged credential chain would yield a forgery in showing, since it includes an issuer that wasn't issued a credential.

all representations of a public key, (*i.e.*, $\forall \mathsf{pk} \in \mathcal{PK}, \mathsf{pk}' \in [\mathsf{pk}] : \mathcal{E}_{\mathcal{PK}}(\mathsf{pk}) = \mathcal{E}_{\mathcal{PK}}(\mathsf{pk}')$). Using this extractor ensures that no adversary can trivially win our anonymity game, and a powerful extractor has been used for a similar purpose in many mercurial signatures papers since their introduction in 2019 [20]. We also use an extractor for credentials, $\mathcal{E}_{\mathsf{cred}}$, which extracts the canonical representations $(\mathsf{pk}_1, \ldots, \mathsf{pk}_L)$ of the public keys in the chain that links a credential to the root issuer.

Definition 14 (Correctness). *A threshold delegatable anonymous credentials scheme* TDAC *(Definition 13) is correct if for all $\lambda \in \mathbb{N}$, $n, t \in O(\lambda)$ with $0 < t \leq n$, $L \in O(\lambda)$ with $L \geq 2$, $\mathsf{pp} \in \mathsf{Setup}(1^\lambda, 1^L, 1^n, 1^t)$, sets $\mathfrak{T}_2, \ldots, \mathfrak{T}_L \subseteq [n]$ of size t, and receiving parties $j_2, \ldots, j_{L-1} \in [n]$, the following holds. Issue a credential down the chain as follows, where the third to fifth lines are repeated for $k = 2, \ldots, L-1$:*

$$
\begin{aligned}
(\mathsf{pk}_1, \{\mathsf{sk}_{1,i}\}_{i\in[n]}) &\leftarrow \mathsf{IssKeyGen}(\mathsf{pp}, 1, \bot),\\
\mathsf{cred}_1 &= (\mathsf{pk}_1, \bot),\\
(\mathsf{pk}_k, \{\mathsf{sk}_{k,i}\}_{i\in[n]}) &\leftarrow \mathsf{IssKeyGen}(\mathsf{pp}, k, \mathsf{pk}_{k-1}),\\
\mathsf{pcred}_{k,i} &\leftarrow \langle \mathsf{Issue}(\mathsf{sk}_{k-1,i}, \mathsf{cred}_{k-1}, k) \leftrightarrow \mathsf{Receive}(\mathsf{sk}_{k,j_k}, k)\rangle, \quad i \in \mathfrak{T}_k,\\
\mathsf{cred}_k &\leftarrow \mathsf{CredComb}(\{\mathsf{pcred}_{k,i}\}_{i\in\mathfrak{T}_k}, \mathsf{pk}_k),\\
(\mathsf{pk}, \mathsf{sk}) &\leftarrow \mathsf{UserKeyGen}(\mathsf{pp}),\\
\mathsf{pcred}_{L,i} &\leftarrow \langle \mathsf{UIssue}(\mathsf{sk}_{L-1,i}, \mathsf{cred}_{L-1}) \leftrightarrow \mathsf{UReceive}(\mathsf{sk})\rangle, \quad i \in \mathfrak{T}_L,\\
\mathsf{cred}_L &\leftarrow \mathsf{UCredComb}(\{\mathsf{pcred}_{L,i}\}_{i\in\mathfrak{T}_L}, \mathsf{pk}).
\end{aligned}
$$

Then $\Pr[\,\langle \mathsf{Prove}(\mathsf{sk}, \mathsf{cred}_L) \leftrightarrow \mathsf{Verify}(\mathsf{pk}_1)\rangle = 1\,] = 1$

Anonymity. We define anonymity in Definition 15, using the game ANON in Figure 8. Anonymity means that a showing reveals nothing about the user beyond the root of its credential chain: neither who the user is nor through which intermediate issuers the credential was issued. This must hold even against an adversary that controls the root authority and up to $t-1$ parties of every honest issuer. The game proceeds as follows.

1. *Root.* The challenger generates the public parameters pp, and the adversary outputs the root public key pk; it may know the root's secret key. In this game, we overload Setup so that it also outputs a trapdoor td, which is never given to the adversary. The trapdoor is only an input to the extractors (step 3): for a scheme whose setup has a trapdoor, they can then be efficient algorithms instead of unbounded ones. Our construction samples no secrets in Setup, so $\mathsf{td} = \bot$.
2. *Learning phase.* The adversary uses the oracles of Figure 7. It creates honest issuers and users and has them issued credentials, either by its own issuers ($\mathcal{O}^{\mathsf{IssueFromAdv}}$, $\mathcal{O}^{\mathsf{UIssueFromAdv}}$) or by honest issuers ($\mathcal{O}^{\mathsf{CreateHI}}$, which also gives it the key shares of $t-1$ parties of the new issuer, and $\mathcal{O}^{\mathsf{CreateHU}}$). It can obtain credentials from honest issuers ($\mathcal{O}^{\mathsf{IssueToAdv}}$, $\mathcal{O}^{\mathsf{UIssueToAdv}}$) and see showings of honest users ($\mathcal{O}^{\mathsf{ProveToAdv}}$).

$\mathcal{O}^{\mathsf{CreateHI}}(\mathsf{id_I}, \mathfrak{T}_\mathsf{I}, \mathsf{id_R}, \mathfrak{T}_\mathsf{Adv}, j, L')$

1 : **if** $\mathsf{SK}[\mathsf{id_I}] = \bot$ // Handle $\mathsf{id_I}$ doesn't exist
$\vee\ \mathsf{SK}[\mathsf{id_R}] \neq \bot$ // Handle $\mathsf{id_R}$ already exists
$\vee\ |\mathfrak{T}_\mathsf{Adv}| \neq t-1 \vee |\mathfrak{T}_\mathsf{I}| \neq t$
$\vee\ L' \geq L$
$\vee\ |\mathsf{CRED}[\mathsf{id_I}]| \neq L'-1$ **then**
2 : **return** $\bot$
// Generate the new issuer's keys and update maps:
3 : $\mathsf{pk}' \leftarrow$ last key in $\mathsf{CRED}[\mathsf{id_I}]$
4 : $(\mathsf{pk_R}, \{\mathsf{sk}_{\mathsf{R},i}\}_{i\in[n]}) \leftarrow \mathsf{IssKeyGen}(\mathsf{pp}, L', \mathsf{pk}')$
5 : $\mathsf{SK}[\mathsf{id_R}] \leftarrow \{\mathsf{sk}_{\mathsf{R},i}\}_{i\in[n]}$
6 : $\mathsf{PK}[\mathsf{id_R}] \leftarrow \mathcal{E}_{\mathcal{PK}}(\mathsf{pk_R})$
// Issue the new issuer's credential:
7 : $\{\mathsf{pcred}_i\}_{i\in\mathfrak{T}_\mathsf{I}} \leftarrow \left\langle \begin{matrix} \mathsf{Issue}(\mathsf{SK}[\mathsf{id_I}]_i, \mathsf{CRED}[\mathsf{id_I}], L') \\ \leftrightarrow \\ \mathsf{Receive}(\mathsf{sk}_{\mathsf{R},j}, L') \end{matrix} \right\rangle_{i\in\mathfrak{T}_\mathsf{I}}$
8 : $\mathsf{cred} \leftarrow \mathsf{CredComb}(\{\mathsf{pcred}_i\}_{i\in\mathfrak{T}_\mathsf{I}}, \mathsf{pk_R})$
9 : $\mathsf{CRED}[\mathsf{id_R}] \leftarrow \mathsf{cred}$
10 : **return** $\mathsf{pk_R}, \{\mathsf{sk}_{\mathsf{R},i}\}_{i\in\mathfrak{T}_\mathsf{Adv}}$

$\mathcal{O}^{\mathsf{CreateHU}}(\mathsf{id_I}, \mathfrak{T}_\mathsf{I}, \mathsf{id_R}) \to \mathsf{pk}$

1 : Same as $\mathcal{O}^{\mathsf{CreateHI}}$ with the following changes:
2 : $(\mathsf{pk_R}, \mathsf{sk_R}) \leftarrow \mathsf{UserKeyGen}(\mathsf{pp})$ instead of $\mathsf{IssKeyGen}$,
3 : and $\mathsf{SK}[\mathsf{id_R}] \leftarrow \mathsf{sk_R}$;
4 : UIssue, $\mathsf{UReceive}(\mathsf{sk_R})$, $\mathsf{UCredComb}$
5 : instead of Issue, $\mathsf{Receive}$, $\mathsf{CredComb}$;
6 : $L' = L$ (no check $L' \geq L$);
7 : $\mathfrak{T}_\mathsf{Adv} = \emptyset$;
8 : it returns only $\mathsf{pk_R}$.

$\mathcal{O}^{\mathsf{ProveToAdv}}(\mathsf{id_P}) \leftrightarrow \mathcal{A}$

1 : **if** $\mathsf{SK}[\mathsf{id_P}] = \bot \vee \mathsf{CRED}[\mathsf{id_P}] = \bot$
$\vee\ |\mathsf{CRED}[\mathsf{id_P}]| \neq L$ **then return** $\bot$
2 : $\mathsf{Prove}(\mathsf{SK}[\mathsf{id_P}], \mathsf{CRED}[\mathsf{id_P}]) \leftrightarrow \mathcal{A}$
3 : **return** $\bot$

$\mathcal{O}^{\mathsf{IssueToAdv}}(\mathsf{id_I}, \mathfrak{T}_\mathsf{I}) \leftrightarrow \mathcal{A}$

1 : **if** $\mathsf{SK}[\mathsf{id_I}] = \bot$ **then return** $\bot$
2 : **if** $\mathsf{CRED}[\mathsf{id_I}] = \bot$ **then return** $\bot$
3 : **if** $|\mathfrak{T}_\mathsf{I}| \neq t$ **then return** $\bot$
4 : $L' \leftarrow |\mathsf{CRED}[\mathsf{id_I}]| + 1$
5 : **if** $L' \geq L$ **then return** $\bot$
6 : $\left\langle \begin{matrix} \mathsf{Issue}(\mathsf{SK}[\mathsf{id_I}]_i, \mathsf{CRED}[\mathsf{id_I}], L') \\ \leftrightarrow \\ \mathcal{A} \end{matrix} \right\rangle_{i\in\mathfrak{T}_\mathsf{I}}$
7 : **return** $\bot$

$\mathcal{O}^{\mathsf{IssueFromAdv}}(\mathsf{id_R}, \mathsf{pk}', \mathfrak{T}_\mathsf{R}, j, L') \leftrightarrow \mathcal{A}$

1 : **if** $\mathsf{SK}[\mathsf{id_R}] \neq \bot$
$\vee\ L' \notin [2, L-1]$
$\vee\ |\mathfrak{T}_\mathsf{R}| \neq t$ **then return** $\bot$
2 : $(\mathsf{pk_R}, \{\mathsf{sk}_{\mathsf{R},i}\}_{i\in[n]}) \leftarrow \mathsf{IssKeyGen}(\mathsf{pp}, L', \mathsf{pk}')$
3 : $\mathsf{SK}[\mathsf{id_R}] \leftarrow \{\mathsf{sk}_{\mathsf{R},i}\}_{i\in[n]}$
4 : $\mathsf{PK}[\mathsf{id_R}] \leftarrow \mathcal{E}_{\mathcal{PK}}(\mathsf{pk_R})$
5 : $\{\mathsf{pcred}_i\}_{i\in\mathfrak{T}_\mathsf{R}} \leftarrow \left\langle \begin{matrix} \mathcal{A}(\mathsf{pk_R}, i) \\ \leftrightarrow \\ \mathsf{Receive}(\mathsf{SK}[\mathsf{id_R}]_j, L') \end{matrix} \right\rangle_{i\in\mathfrak{T}_\mathsf{R}}$
6 : $\mathsf{cred} \leftarrow \mathsf{CredComb}(\{\mathsf{pcred}_i\}_{i\in\mathfrak{T}_\mathsf{R}}, \mathsf{pk_R})$
7 : $\mathsf{CRED}[\mathsf{id_R}] \leftarrow \mathsf{cred}$
8 : **return** $\bot$

$\mathcal{O}^{\mathsf{UIssueFromAdv}}(\mathsf{id_R}, \mathfrak{T}_\mathsf{R}) \leftrightarrow \mathcal{A}$

1 : Same as $\mathcal{O}^{\mathsf{IssueFromAdv}}$ with the following changes:
2 : $(\mathsf{pk_R}, \mathsf{sk_R}) \leftarrow \mathsf{UserKeyGen}(\mathsf{pp})$
3 : instead of $\mathsf{IssKeyGen}$, and $\mathsf{SK}[\mathsf{id_R}] \leftarrow \mathsf{sk_R}$;
4 : $\mathsf{UReceive}(\mathsf{sk_R})$ instead of $\mathsf{Receive}$,
5 : $\mathsf{UCredComb}$ instead of $\mathsf{CredComb}$;
6 : and $L' = L$ (no check on L').

$\mathcal{O}^{\mathsf{UIssueToAdv}}(\mathsf{id_I}, \mathfrak{T}_\mathsf{I}) \leftrightarrow \mathcal{A}$

1 : Same as $\mathcal{O}^{\mathsf{IssueToAdv}}$ with the following changes:
2 : UIssue instead of Issue,
3 : and require $L' = L$ instead
4 : of $L' < L$.

Fig. 7: Definition of Oracles for TDAC scheme (defined in Definition 13). All party sets must be subsets of $[n]$ and $j \in [n]$; otherwise, an oracle returns $\bot$.

3. *Challenge.* The adversary names two honest users id_0 and id_1. Their credential chains may differ, but the challenger checks that both chains start at pk, that all their other keys belong to honest issuers or users, and that both credentials verify; if not, the adversary loses. Without these checks, the adversary could win trivially, *e.g.*, by recognizing the key of one of its own issuers in the showing. To compare keys regardless of how they were randomized, the

challenger uses the extractors $\mathcal{E}_{\mathsf{cred}}$ and $\mathcal{E}_{\mathcal{PK}}$, which map all representations of a key to the same canonical one.
4. *Showing.* The challenger picks a random bit b, and user id_b shows its credential to the adversary.
5. *Guess.* After further oracle queries, the adversary outputs a bit b' and wins if $b' = b$.

The scheme is anonymous if no PPT adversary wins with probability non-negligibly greater than 1/2.

Definition 15 (Anonymity). *A threshold delegatable anonymous credentials scheme* TDAC *(Definition 13) is anonymous if there exist (possibly unbounded) extractors $\mathcal{E}_{\mathcal{PK}}$ and $\mathcal{E}_{\mathsf{cred}}$, which may also take* td *as input (we omit it from the notation), such that for all $\lambda \in \mathbb{N}$, $n, t \in O(\lambda)$ with $0 < t \leq n$, $L \in O(\lambda)$, and all* PPT *adversaries $\mathcal{A}$ with access to all oracles in Figure 7,*

$$\mathsf{Adv}^{\mathsf{ANON}}_{\mathsf{TDAC},\mathcal{A}}(\lambda) = \Pr\left[\mathsf{ANON}^{\mathsf{TDAC}}_{\mathcal{A}}(1^\lambda) = 1\right] - \frac{1}{2} \leq \mathsf{negl}(\lambda),$$

where the game $\mathsf{ANON}^{\mathsf{TDAC}}_{\mathcal{A}}$ *is defined in Figure 8.*

$\mathsf{ANON}^{\mathsf{TDAC}}_{\mathcal{A}}(1^\lambda)$

$(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, 1^L, 1^n, 1^t)$
$(\mathsf{pk}, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$
$(\mathsf{id}_0, \mathsf{id}_1, \mathsf{st}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{st})$
for $c \in \{0,1\}$ **do**
  if $|\mathsf{CRED}[\mathsf{id}_c]| \neq L$: **return** 0
  $(\mathsf{pk}_{c,k})_{k\in[L]} \leftarrow \mathcal{E}_{\mathsf{cred}}(\mathsf{CRED}[\mathsf{id}_c])$
  if $\mathsf{pk}_{c,1} \neq \mathcal{E}_{\mathcal{PK}}(\mathsf{pk})$: **return** 0
  if $\mathsf{pk}_{c,k} \notin \mathsf{PK}$ for some $k \geq 2$: **return** 0
  $d \leftarrow \langle \mathsf{Prove}(\mathsf{SK}[\mathsf{id}_c], \mathsf{CRED}[\mathsf{id}_c]) \leftrightarrow \mathsf{Verify}(\mathsf{pk})\rangle$
  if $d = 0$: **return** 0
$b \xleftarrow{\$} \{0,1\}$
$\mathsf{st} \leftarrow \langle \mathsf{Prove}(\mathsf{SK}[\mathsf{id}_b], \mathsf{CRED}[\mathsf{id}_b]) \leftrightarrow \mathcal{A}(\mathsf{st})\rangle$
$b' \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{st})$
return $b = b'$

$\mathsf{unForge}^{\mathsf{TDAC}}_{\mathcal{A}}(1^\lambda)$

$(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, 1^L, 1^n, 1^t)$
$(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{IssKeyGen}(\mathsf{pp}, 1, \bot)$
$\mathsf{SK}[0] \leftarrow \{\mathsf{sk}_i\}_{i\in[n]}$
$\mathsf{CRED}[0] \leftarrow (\mathsf{pk}, \bot)$; $\mathsf{PK}[0] \leftarrow \mathcal{E}_{\mathcal{PK}}(\mathsf{pk})$
$\mathsf{st} \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Unf}}}(\mathsf{pp}, \mathsf{pk})$
$b \leftarrow \langle \mathcal{A}(\mathsf{st}) \leftrightarrow \mathsf{Verify}(\mathsf{pk})\rangle$
return b

Fig. 8: Anonymity game $\mathsf{ANON}^{\mathsf{TDAC}}_{\mathcal{A}}(1^\lambda)$ and unforgeability game $\mathsf{unForge}^{\mathsf{TDAC}}_{\mathcal{A}}(1^\lambda)$ for a TDAC scheme. The oracles $\mathcal{O}$ are those of Figure 7; $\mathcal{O}_{\mathsf{Unf}}$ consists of $\mathcal{O}^{\mathsf{CreateHI}}$, $\mathcal{O}^{\mathsf{CreateHU}}$, and $\mathcal{O}^{\mathsf{ProveToAdv}}$. We write $\mathsf{pk} \in \mathsf{PK}$ if $\mathsf{pk} = \mathsf{PK}[\mathsf{id}]$ for some handle id. In both games, Setup is overloaded to also output a trapdoor td, which is never given to the adversary; it is only an input to the extractors (Appendix E.1).

Unforgeability. We define unforgeability for our threshold DAC scheme in Definition 16. We define unforgeability in Definition 16, using the game unForge

in Figure 8. Unforgeability means that no one can show a credential without having been issued one, even after seeing showings by honest users and while holding the key shares of up to $t-1$ parties of every honest intermediate issuer. The game proceeds as follows.

1. *Root.* The challenger generates the public parameters pp and an honest threshold root key pk, which it registers under the handle $\mathsf{id} = 0$; the adversary receives pp and pk. As in the anonymity game, Setup is overloaded to also output a trapdoor td, which is never given to the adversary; here it is only an input to the extractor that records honest keys in PK.
2. *Learning phase.* The adversary may only use the oracles through which honest parties obtain credentials. It creates honest intermediate issuers below the root or below other honest issuers ($\mathcal{O}^{\mathsf{CreateHI}}$, which also gives it the key shares of $t-1$ parties of the new issuer), creates honest users ($\mathcal{O}^{\mathsf{CreateHU}}$), and sees showings of honest users ($\mathcal{O}^{\mathsf{ProveToAdv}}$). The other oracles are excluded, since with them the adversary could simply be issued a credential (*e.g.*, via $\mathcal{O}^{\mathsf{UIssueToAdv}}$) and show it.
3. *Forgery.* The adversary runs the showing protocol with an honest verifier for pk and wins if the verifier accepts. It keeps its state from the learning phase but has no oracle access during this showing, so it cannot relay an honest user's showing ($\mathcal{O}^{\mathsf{ProveToAdv}}$) to the verifier.

Since the adversary is never issued a credential, every accepted showing is a forgery. The scheme is unforgeable if no PPT adversary wins with non-negligible probability.

Definition 16 (Unforgeability). *A threshold delegatable anonymous credentials scheme* TDAC *(Definition 13) is unforgeable if for all* $\lambda \in \mathbb{N}$, $n, t \in O(\lambda)$ *with* $0 < t \leq n$, $L \in O(\lambda)$, *and all* PPT *adversaries* $\mathcal{A}$ *with access to the oracles* $\mathcal{O}_{\mathsf{Unf}} = \{\mathcal{O}^{\mathsf{CreateHI}}, \mathcal{O}^{\mathsf{CreateHU}}, \mathcal{O}^{\mathsf{ProveToAdv}}\}$ *in Figure 7,*

$$\mathsf{Adv}^{\mathsf{unForge}}_{\mathsf{TDAC},\mathcal{A}}(\lambda) = \Pr\left[\,\mathsf{unForge}^{\mathsf{TDAC}}_{\mathcal{A}}(1^\lambda) = 1\,\right] \leq \mathsf{negl}(\lambda),$$

where the game $\mathsf{unForge}^{\mathsf{TDAC}}_{\mathcal{A}}$ *is defined in Figure 8.*

E.2 Construction of TDAC

In Figure 9, we present a construction of our threshold delegatable anonymous credentials scheme $\mathsf{TDAC} = (\mathsf{Setup}, \mathsf{IssKeyGen}, \mathsf{UserKeyGen}, \mathsf{Issue}, \mathsf{Receive}, \mathsf{CredComb}, \mathsf{UIssue}, \mathsf{UReceive}, \mathsf{UCredComb}, \mathsf{Prove}, \mathsf{Verify})$ (Definition 13), which we call $\mathbf{\Pi}_{\mathsf{TDAC}}$; Section 4 gives an overview.

Parameters. A security parameter $\lambda \in \mathbb{N}$, the length $L \geq 2$ of credential chains, the number n of parties that share each issuer key, and the threshold $t \leq n$.

Building blocks. A leveled cross-scheme correct threshold mercurial signature scheme TMS (Definitions 3 and 8); a non-interactive zero-knowledge proof of

knowledge NIZK for issuance (*e.g.*, Fiat–Shamir applied to a Chaum–Pedersen proof); and an interactive zero-knowledge proof of knowledge of discrete logarithms [16] for showing.

The secret key sk_i of each issuer party contains, besides its key share, the issuer's public key and its encoding enc as a message for the issuer above, which the dealer computes in IssKeyGen. A user's public key is signed into its credential, and Prove includes a proof of knowledge of the corresponding secret key, so a credential cannot be shown without that key. This gives *non-transferability* [15] (Section 1.1).

E.3 Security Analysis of TDAC

We formally state the security of our TDAC scheme in Figure 9 in Theorems 8 to 10, and prove each right after its statement.

Theorem 8 (Correctness). *Assume that* TMS *is correct (Definition 4) and leveled cross-scheme correct (Definition 8), and that NIZK and the proof of knowledge in* Prove *are complete. Then our* $\mathbf{\Pi}_{\mathsf{TDAC}}$ *construction in Figure 9 is a correct threshold delegatable anonymous credentials scheme as per Definition 14.*

Proof (Proof of Theorem 8). Fix the parameters, the sets $\mathfrak{T}_k$, and the receiving parties j_k as in Definition 14. Assume that no tag secret sampled by GenTagMsg has a zero entry and that no hash value h equals 1; both hold except with negligible probability. We first show, by induction on $k \in [2, L]$, that $\mathsf{cred}_k = \mathsf{cred}_{k-1} \| (\mathsf{pk}_k, \vec{T}_k, \vec{M}_k, \sigma_k)$ with

$$\mathsf{TMS.Verify}_{\mathsf{pp}_{k-1}}(\mathsf{pk}_{k-1}, \vec{T}_k, (\vec{M}_k, \mathsf{pk}_k), \sigma_k) = 1,$$

where $\mathsf{pk}_L = \mathsf{pk}$ and $\mathsf{cred}_1 = (\mathsf{pk}_1, \perp)$.

Issuance. For $k < L$, the dealer in $\mathsf{IssKeyGen}(\mathsf{pp}, k, \mathsf{pk}_{k-1})$ obtains $(\vec{\rho}, \vec{T}_k, (\vec{M}_k, \vec{N})) \leftarrow \mathsf{TMS.GenTagMsg}_{\mathsf{pp}_{k-1}}(\mathsf{sk}_0, \mathsf{pk}_{k-1})$, and $\vec{N} = \mathsf{pk}_k$ by leveled cross-scheme correctness (Definition 8). For $k = L$, the user does the same with its secret key sk and obtains $\vec{N} = \hat{P}^{\mathsf{sk}} = \mathsf{pk}$. In Issue (or UIssue), each issuer party $i \in \mathfrak{T}_k$ takes pk_I to be the last key of cred_{k-1}, which is pk_{k-1}, so it recomputes the hash value h used by GenTagMsg. Its checks pass: π verifies by completeness of NIZK; $R_j \neq 1$ since $\rho_j \neq 0$; and $e(M_j, \hat{P}) = e(T_j, \hat{N}_j)$ since $M_j = T_j^{m_j}$ and $\hat{N}_j = \hat{P}^{m_j}$ for the message secret $\vec{m}$. Since σ_i depends on $\vec{\rho}$ only through h and $\vec{T}_k$, it equals $\mathsf{TMS.ParSign}_{\mathsf{pp}_{k-1}}(\mathsf{sk}_{k-1,i}, \mathsf{pk}_{k-1}, \vec{\rho}, (\vec{M}_k, \mathsf{pk}_k))$. By partial and threshold verification (Definition 4), TMS.Reconst in CredComb (or UCredComb) accepts these partial signatures and outputs a signature σ_k as required.

Showing. Let $\mathsf{cred} = \mathsf{cred}_L$, and let $k \in [2, L]$. By key and signature conversion (Definition 4), σ_k^* verifies on $(\vec{T}_k, (\vec{M}_k, \mathsf{pk}_k))$ under $\mathsf{pk}'_{k-1} = \mathsf{TMS.ConvertPK}(\mathsf{pk}_{k-1}; \omega_{k-1})$; for $k = 2$, $\omega_1 = 1$ and $\mathsf{pk}'_1 = \mathsf{pk}_1$. By change of representative, applied to pk'_{k-1} (which is itself a possible output of KeyGen,

with signing key $\omega_{k-1}\mathsf{sk}$), $(\vec{T}'_k, (\vec{M}'_k, \mathsf{pk}'_k), \sigma'_k)$ then verifies under pk'_{k-1}, where $\mathsf{pk}'_k = \mathsf{pk}_k^{\omega_k}$ and $\vec{T}'_k = \vec{T}_k^{\gamma_k}$; no entry of $\vec{T}'_k$ is 1, since $T_{k,j} = h^{\rho_j} \neq 1$. Hence the chain starts at pk_1 and all of the verifier's checks on it pass. Finally, $\mathsf{pk}'_L = \mathsf{pk}^{\omega_L} = \hat{P}^{\omega_L \mathsf{sk}}$, so the prover knows $\mathsf{sk}' = \omega_L \mathsf{sk}$, and the proof of knowledge accepts by completeness. Therefore $\langle \mathsf{Prove}(\mathsf{sk}, \mathsf{cred}_L) \leftrightarrow \mathsf{Verify}(\mathsf{pk}_1) \rangle = 1$.

Theorem 9 (Unforgeability). *In the random oracle model, assume that the cross-scheme correct version of* TMS *is unforgeable (Definition 5), that the symmetric discrete logarithm problem (given $(P^a, \hat{P}^a)$ for $a \xleftarrow{\$} \mathbb{Z}_p^\times$, compute a) is hard, that* NIZK *is zero-knowledge, and that the proof of knowledge in* Prove *is zero-knowledge and knowledge-sound. Then our $\mathbf{\Pi}_{\mathsf{TDAC}}$ construction in Figure 9 is an unforgeable threshold delegatable anonymous credentials scheme* TDAC *as per Definition 16.*

Proof (Proof of Theorem 9). Let $\mathcal{A}$ be a PPT adversary in $\mathsf{unForge}_{\mathcal{A}}^{\mathsf{TDAC}}$ that makes at most q oracle queries, and suppose that the verifier accepts its final showing, in which $\mathcal{A}$ sends a chain $\mathsf{cred}' = (\mathsf{pk}_1, \perp) \| (\mathsf{pk}'_k, \vec{T}'_k, \vec{M}'_k, \sigma'_k)_{k=2}^L$ and runs the proof of knowledge. Since $\mathcal{A}$ has no oracle access during this showing, we can run the knowledge extractor of the proof of knowledge on $\mathcal{A}$ in this stage; by knowledge soundness, it outputs sk' with $\mathsf{pk}'_L = \hat{P}^{\mathsf{sk}'}$ with probability negligibly close to the probability that the verifier accepts.

We call a level $k \in [2, L]$ *fresh* if pk'_{k-1} is in the class of the key of an honest issuer id, *i.e.*, $\mathcal{E}_{\mathcal{PK}}(\mathsf{pk}'_{k-1}) = \mathsf{PK}[\mathsf{id}]$, but $(\vec{M}'_k, \mathsf{pk}'_k)$ is not in the class of any message on which a party of id produced a partial signature. Since $\mathsf{pk}'_1 = \mathsf{pk}$ is the honest root key and, in this game, honest issuers issue credentials only to honest parties, induction over k shows that if no level is fresh, then every pk'_k is in the class of the key of an honest party; in particular, pk'_L is in the class of the key pk of an honest user. We bound the two cases separately.

Case 1: some level k is fresh. We build $\mathcal{B}$ against the unforgeability of TMS (Definition 5). $\mathcal{B}$ first guesses the level k, since it must fix the parameters of its challenger, which are those of level $k-1$, before the game starts; it builds the parameters of all other levels from the same bilinear group and hash function, as ExtendSetup only changes the message length. It then guesses the handle id of the issuer at level $k-1$ (the root or one of the at most q issuers created by $\mathcal{O}^{\mathsf{CreateHI}}$). Both guesses are correct with probability at least $1/((q+1)L)$. When id is created, $\mathcal{B}$ sends the corrupted set $\mathfrak{T}_{\mathsf{Adv}}$ and the key of id's parent to its challenger and uses the challenge key as the key of id; the $t-1$ shares that $\mathcal{O}^{\mathsf{CreateHI}}$ returns are those of the corrupted parties. $\mathcal{B}$ generates all other keys itself, including those of all receivers. Hence it knows the tag secret of every message that id signs and obtains each partial signature of id from $\mathcal{O}_{\mathsf{PSign}}$; the issuance proof for id's own encoding it simulates with the simulator of NIZK. For $k \geq 3$, the parent of id is an honest issuer whose key $\mathcal{B}$ knows, so $\mathcal{B}$ computes the parent's signature on id's encoding itself, with a uniform first component; the adversary does not see this issuance. $\mathcal{B}$ answers $\mathcal{O}^{\mathsf{ProveToAdv}}$ honestly, since it knows all user keys and credentials. Finally, it outputs $(\vec{T}'_k, (\vec{M}'_k, \mathsf{pk}'_k), \sigma'_k)$, which verifies under pk'_{k-1}; since level k is fresh, this message class was never queried.

Case 2: no level is fresh. Then $\mathsf{pk}'_L = \mathsf{pk}^\omega$ for some $\omega \in \mathbb{Z}_p^\times$ and the key pk of an honest user, and the extracted key is $\mathsf{sk}' = \omega\mathsf{sk}$. We build $\mathcal{B}'$ that, given $(P^a, \hat{P}^a)$, computes a. $\mathcal{B}'$ guesses the user (one of at most q), samples $r \xleftarrow{\$} \mathbb{Z}_p^\times$, and sets this user's key to $\mathsf{pk} = (\hat{P}^a, \hat{P}^r)$. It programs the random oracle so that the hash value h of this user's encoding is P^η for a known η, which lets it compute $M_1 = T_1^a = (P^a)^{\eta\rho_1}$ from P^a; it computes this user's issuance proof honestly, since it chose $\vec{\rho}$, simulates the user's proofs of knowledge in $\mathcal{O}^{\mathsf{ProveToAdv}}$ with the zero-knowledge simulator, and runs everything else honestly. From $\mathsf{sk}' = (\omega a, \omega r)$, $\mathcal{B}'$ computes $\omega = \mathsf{sk}'_2/r$ and then $a = \mathsf{sk}'_1/\omega$.

Hence the probability that $\mathcal{A}$ wins is at most $(q+1)L \cdot \mathsf{Adv}_{\mathcal{B}} + q \cdot \mathsf{Adv}_{\mathcal{B}'}$ plus negligible terms, which is negligible.

Theorem 10 (Anonymity). *If the cross-scheme correct version of* TMS *is public key class-hiding (Definition 6), message and tag class-hiding (Definition 6), and origin-hiding (Definition 7), and the issuance NIZK and the proof of knowledge in* Prove *are zero-knowledge, then our* $\mathbf{\Pi}_{\mathsf{TDAC}}$ *construction in Figure 9 is an anonymous threshold delegatable anonymous credentials scheme as per Definition 15.*

Proof (Proof of Theorem 10). Let $\mathcal{A}$ be a PPT adversary in $\mathsf{ANON}_{\mathcal{A}}^{\mathsf{TDAC}}$, and let G_c denote this game with the challenge bit fixed to $b = c$. If a check before the challenge fails, the game returns 0 for both values of b, so it suffices to show that $\mathcal{A}$'s view of the challenge showing is computationally indistinguishable in G_0 and G_1 when all checks pass. In that case, both credentials verify and have length L; both chains start at pk itself, since the verifier checks that the first key equals pk; and all other keys in both chains belong to honest issuers (levels 2 to $L-1$) or to the two honest users (level L). The challenge showing consists of $\mathsf{cred}' = (\mathsf{pk}, \bot) \| (\mathsf{pk}'_k, \vec{T}'_k, \vec{M}'_k, \sigma'_k)_{k=2}^L$ and a proof of knowledge. For each $c \in \{0,1\}$, we change G_c in the following steps, until the showing no longer depends on c.

1. *Simulated proofs.* All proofs of knowledge of the two challenge users (in the challenge showing and in $\mathcal{O}^{\mathsf{ProveToAdv}}$), and the issuance proofs π that honest receivers send to the adversary's issuers ($\mathcal{O}^{\mathsf{IssueFromAdv}}$, $\mathcal{O}^{\mathsf{UIssueFromAdv}}$), are produced by the zero-knowledge simulators. This changes $\mathcal{A}$'s view negligibly, since both proof systems are zero-knowledge.
2. *Signatures.* Each σ'_k in the challenge showing is replaced by the signature with a fresh uniform first component h'_k that verifies on $(\vec{T}'_k, (\vec{M}'_k, \mathsf{pk}'_k))$ under pk'_{k-1}; it is well defined, since the other two components are determined by h'_k, the tag, the message, and the key (Figure 3).
3. *Issuer keys.* For $k = 2, \ldots, L-1$ in turn, the key pk'_k of the level-k issuer of chain c and its encoding $(\vec{T}'_k, \vec{M}'_k)$ are replaced by a fresh key and its encoding, independent of all other keys. Each step is indistinguishable by public key class-hiding of TMS (Definition 6): the reduction embeds this issuer's key as key 1 of the game, with the issuer's corrupted parties as $\mathcal{C}$ and its parent's key as pk', answers the issuer's partial signatures with $\mathcal{O}_{\mathsf{PSign}_1}$, uses key 2

and its encoding in the challenge showing, and obtains the signature on the next level's encoding with $\mathcal{O}_{\mathsf{Sign}_2^b}$. If the issuer's parent is honest, the reduction computes the parent's signature on the issuer's encoding itself, with a uniform first component; this is possible since it knows the parent's key, and the adversary does not see this issuance. All other showings, *e.g.*, of other users below this issuer, the reduction computes itself, since ConvertPK and ConvertSig need no secret. For $b = 1$, key 2 and its encoding are distributed exactly as in the real showing (ConvertPK and ChangeRep with fresh randomness); for $b = 0$, they are fresh.

4. *User.* The user's encoding $(\vec{T}'_L, (\vec{M}'_L, \mathsf{pk}'_L))$ is the output of ChangeRep with fresh (γ_L, ω_L), so by origin-hiding (Definition 7) it is uniform in the tag and message classes of the user's encoding. It is replaced by a freshly sampled tag and message, which is indistinguishable by tag and message class-hiding (Definition 6). The reduction computes the user's other showings from further randomizations of the given encoding: it knows the key of the honest issuer at level $L - 1$, so it can sign them itself, and their proofs of knowledge are simulated (step 1).

After these steps, the challenge showing consists of pk, fresh keys, tags, and messages, signatures with fresh first components, and a simulated proof; its distribution does not depend on c. Hence G_0 and G_1 are computationally indistinguishable, and the advantage of $\mathcal{A}$ is negligible.

$\mathsf{Setup}(1^\lambda, 1^L, 1^n, 1^t)$

1 : $\mathsf{pp}_{L-1} \leftarrow \mathsf{TMS.Setup}(1^\lambda, 1^2)$ // user keys have length 2
2 : **for** $k = L-1, \ldots, 2$: $\mathsf{pp}_{k-1} \leftarrow \mathsf{TMS.ExtendSetup}(\mathsf{pp}_k)$
// keys under pp_k are messages under pp_{k-1}
3 : **return** $\mathsf{pp} = ((\mathsf{pp}_k)_{k\in[L-1]}, n, t)$

$\mathsf{IssKeyGen}(\mathsf{pp}, L', \mathsf{pk}')$

1 : $(\vec{\mathsf{sk}}, \vec{\mathsf{pk}}, \mathsf{pk}) \leftarrow \mathsf{TMS.KeyGen}_{\mathsf{pp}_{L'}}(n, t)$ // $\vec{\mathsf{sk}}$ shares sk_0
2 : $\mathsf{enc} \leftarrow \bot$
3 : **if** $L' > 1$: // encode pk as a message for pk'
4 : $(\vec{\rho}, \vec{T}, (\vec{M}, \mathsf{pk})) \leftarrow \mathsf{TMS.GenTagMsg}_{\mathsf{pp}_{L'-1}}(\mathsf{sk}_0, \mathsf{pk}')$
5 : $\vec{R} = (P^{\rho_j})_j$; $h = \mathsf{H}(\vec{R}\|\mathsf{pk}\|\mathsf{pk}')$
6 : $\pi \leftarrow \mathsf{NIZK}\{\vec{\rho} : \forall j : R_j = P^{\rho_j} \wedge T_j = h^{\rho_j}\}$
7 : $\mathsf{enc} \leftarrow (\vec{R}, \vec{T}, \vec{M}, \pi)$
8 : erase sk_0 and $\vec{\rho}$
9 : **return** pk, $\{\mathsf{sk}_i = (\mathsf{sk}_i, \mathsf{pk}_i, \mathsf{pk}, \mathsf{enc})\}_{i\in[n]}$

$\mathsf{UserKeyGen}(\mathsf{pp})$

1 : $\mathsf{sk} = (m_1, m_2) \xleftarrow{\$} (\mathbb{Z}_p^\times)^2$
2 : **return** $(\mathsf{pk} = (\hat{P}^{m_1}, \hat{P}^{m_2}), \mathsf{sk})$

$\mathsf{CredComb}(\{\mathsf{pcred}_{\mathsf{R},i}\}_{i\in\mathfrak{T}}, \mathsf{pk}_\mathsf{R})$

1 : parse $\mathsf{pcred}_{\mathsf{R},i} = (\mathsf{cred}_\mathsf{I}, \mathsf{pk}_i, \vec{T}, \vec{M}, \sigma_i)$
2 : $L' \leftarrow |\mathsf{cred}_\mathsf{I}| + 1$ // level of the receiver
3 : $\sigma \leftarrow \mathsf{TMS.Reconst}_{\mathsf{pp}_{L'-1}}((\mathsf{pk}_i)_{i\in\mathfrak{T}}, \vec{T}, (\vec{M}, \mathsf{pk}_\mathsf{R}), \{(i, \sigma_i)\}_{i\in\mathfrak{T}})$
4 : **if** $\sigma = \bot$: **return** $\bot$
5 : **return** $\mathsf{cred}_\mathsf{I}\|(\mathsf{pk}_\mathsf{R}, \vec{T}, \vec{M}, \sigma)$

$\mathsf{UCredComb}(\{\mathsf{pcred}_i\}_{i\in\mathfrak{T}}, \mathsf{pk})$

1 : Same as CredComb, with pk in place of pk_R

$\langle \mathsf{UIssue}(\mathsf{sk}_{\mathsf{I},i}, \mathsf{cred}_\mathsf{I}) \leftrightarrow \mathsf{UReceive}(\mathsf{sk})\rangle$

1 : Same as $\langle \mathsf{Issue} \leftrightarrow \mathsf{Receive}\rangle$ with $L' = L$, except that before line 2, the user computes enc itself as in lines 4–7 of IssKeyGen, but with $(\vec{\rho}, \vec{T}, (\vec{M}, \mathsf{pk})) \leftarrow \mathsf{TMS.GenTagMsg}_{\mathsf{pp}_{L-1}}(\mathsf{sk}, \mathsf{pk}_\mathsf{I})$, and then erases $\vec{\rho}$

$\langle \mathsf{Issue}(\mathsf{sk}_{\mathsf{I},i}, \mathsf{cred}_\mathsf{I}, L') \leftrightarrow \mathsf{Receive}(\mathsf{sk}_{\mathsf{R},j}, L')\rangle$

Issuer party i:
1 : send $(\mathsf{cred}_\mathsf{I}, \mathsf{pk}_i)$, where $\mathsf{sk}_{\mathsf{I},i} = (\mathsf{sk}_i, \mathsf{pk}_i, \cdot, \cdot)$
Receiver party j:
2 : send $(\mathsf{pk}_\mathsf{R}, \mathsf{enc})$, where $\mathsf{sk}_{\mathsf{R},j} = (\cdot, \cdot, \mathsf{pk}_\mathsf{R}, \mathsf{enc})$
Issuer party i:
3 : parse $\mathsf{enc} = (\vec{R}, \vec{T}, \vec{M}, \pi)$ and $\mathsf{pk}_\mathsf{R} = (\hat{N}_j)_j$
4 : $\mathsf{pk}_\mathsf{I} \leftarrow$ last key in cred_I; $h = \mathsf{H}(\vec{R}\|\mathsf{pk}_\mathsf{R}\|\mathsf{pk}_\mathsf{I})$
5 : **if** π is invalid for $(\vec{R}, \vec{T}, h)$ $\vee\ \exists j : R_j = 1 \vee e(M_j, \hat{P}) \neq e(T_j, \hat{N}_j)$: **return** $\bot$
6 : parse $\mathsf{sk}_i = (x_i, (y_{i,j})_j, (z_{i,j})_j)$
7 : $\sigma_i = (h, \prod_j T_j^{z_{i,j}}, h^{x_i} \prod_j M_j^{y_{i,j}})$ // as TMS.ParSign, but without $\vec{\rho}$
8 : send σ_i
Receiver party j:
9 : **return** $\mathsf{pcred}_{\mathsf{R},i} = (\mathsf{cred}_\mathsf{I}, \mathsf{pk}_i, \vec{T}, \vec{M}, \sigma_i)$

$\langle \mathsf{Prove}(\mathsf{sk}, \mathsf{cred}) \leftrightarrow \mathsf{Verify}(\mathsf{pk})\rangle$

Prover:
1 : parse $\mathsf{cred} = (\mathsf{pk}_1, \bot)\|(\mathsf{pk}_k, \vec{T}_k, \vec{M}_k, \sigma_k)_{k=2}^L$
2 : $\omega_1 = 1$; $\mathsf{pk}'_1 = \mathsf{pk}_1$ // the root key is not randomized
3 : **for** $k = 2, \ldots, L$:
4 : $\omega_k, \gamma_k \xleftarrow{\$} \mathbb{Z}_p^\times$
5 : $\sigma_k^* \leftarrow \mathsf{TMS.ConvertSig}_{\mathsf{pp}_{k-1}}(\mathsf{pk}_{k-1}, \vec{T}_k, (\vec{M}_k, \mathsf{pk}_k), \sigma_k; \omega_{k-1})$
// σ_k^* verifies under pk'_{k-1}
6 : $(\vec{T}'_k, (\vec{M}'_k, \mathsf{pk}'_k), \sigma'_k) \leftarrow \mathsf{TMS.ChangeRep}_{\mathsf{pp}_{k-1}}(\mathsf{pk}'_{k-1}, \vec{T}_k, (\vec{M}_k, \mathsf{pk}_k), \sigma_k^*; (\gamma_k, \omega_k))$
// $\mathsf{pk}'_k = \mathsf{pk}_k^{\omega_k}$; also rerandomizes h
7 : send $\mathsf{cred}' = (\mathsf{pk}_1, \bot)\|(\mathsf{pk}'_k, \vec{T}'_k, \vec{M}'_k, \sigma'_k)_{k=2}^L$
Prover and verifier:
8 : zero-knowledge proof of knowledge of sk' with $\mathsf{pk}'_L = \hat{P}^{\mathsf{sk}'}$
// the prover knows $\mathsf{sk}' = \omega_L \mathsf{sk}$
Verifier:
9 : **return** 1 iff $\mathsf{pk}_1 = \mathsf{pk}$, the proof accepts, and for all $k \geq 2$: $\mathsf{TMS.Verify}_{\mathsf{pp}_{k-1}}(\mathsf{pk}'_{k-1}, \vec{T}'_k, (\vec{M}'_k, \mathsf{pk}'_k), \sigma'_k) = 1$ and $T'_{k,j} \neq 1$ for all j

Fig. 9: Construction $\mathbf{\Pi}_{\mathsf{TDAC}}$ of threshold delegatable anonymous credentials. IssKeyGen is run by a trusted dealer, and the receiving party forwards the credential output by CredComb to the other receiving parties. $\mathsf{TMS.Alg}_{\mathsf{pp}_k}$ denotes the algorithm TMS.Alg run with parameters pp_k.