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: 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 realize *non-interactive* TMS, where each participant non-interactively computes its contribution to the threshold mercurial signature. Our construction also substantially reduces the overall communication complexity. It uses the mercurial signature scheme of Mir et al. (CCS 2023) as a starting point. Further, we introduce *threshold delegatable anonymous credentials* (TDAC) and use a non-interactive TMS to construct them.

1 Introduction

In the digital era, services that preserve user anonymity are increasingly important. Without them, ordinary people are in danger of surveillance by powerful entities such as governments, corporations, or hackers. A useful tool for providing privacy while maintaining service integrity (*i.e.,* authorizing only certain users) is *anonymous credentials.* They were first described by Chaum [23], first realized by Camenisch and Lysyanskaya [19], and then improved and extended in many subsequent works [15, 17]. Anonymous credentials allow a user to prove in zero-knowledge that they know a signature without supplying it in the clear. They are, seemingly, the perfect answer to legislative efforts towards privacy-preserving digital identity, such as the EU's Digital Identity Wallet regulation (EUDIW) [12, 36]. The EUDIW requires that EU citizens be able to selectively prove certain information about themselves while protecting other details.

For example, an EUDIW holder should be able to prove EU citizenship without revealing which EU country they are a citizen of. A governing body such as the EU may want to delegate its signing power to smaller regions, such as member countries. In this case, it is important to still preserve the privacy of end users' identity attributes, such as which EU country they are from. This challenge is addressed by *delegatable anonymous credentials* (DACs), introduced and first realized by Chase and Lysyanskaya [22]. 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. Users can present their credentials without revealing which delegatee issued them.

DACs can be realized efficiently and modularly with *mercurial signatures*, a scheme introduced by Crites and Lysyanskaya [28] 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 [9, 26, 29, 46, 50, 55].

Threshold signatures [32,33] 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 [34,58]. Thresholdizing a mercurial signatures can strengthen both anonymity and unforgeability [9]. In particular, if we assume non-collusion among the parties holding the thresholdized keys (so no adversary holds more than $t-1$ shares), threshold mercurial signatures can provide a stronger notion of privacy. We review this in more detail in Section 1.3).

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 [9] and used to construct mix-nets in [8]. However, no known TMS scheme is non-interactive; all existing constructions currently require interaction between the signers to produce a signature. This raises the following question:

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

In this work, we answer this question in the affirmative by formally defining and constructing the first such scheme. We then extend it to 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 delegatable anonymous credentials. 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. [50] which is itself based on the structure-preserving signature of Ghadafi [42].
- We further extend the construction in [50] to support the signing of public keys, overcoming a key limitation in the original scheme that prevented its use in DAC. Thus, we achieve the first non-interactive, non-transferable threshold DAC scheme from mercurial signatures.
  Threshold DAC was previously studied by Mir et al. in [51], but their construction requires an extra round during showing (due to commitments needed for proving correct encryption). Moreover, the scheme does not enforce non-transferability—the property preventing users from maliciously sharing credentials—which is an implicit requirement for anonymous credentials [18]. In general, non-transferability is achieved by signing user public keys, enabling users to prove their identity. Since [51] supports only attribute signing, their construction cannot bind credentials to public keys, and thus cannot prevent malicious credential transfer.

Our construction relies on the generic group model (GGM) [57]. Constructing mercurial signatures is known to at least require non-interactive assumptions [11], and many constructions use generic or algebraic group models [9, 28, 29, 46, 50].

In Table 1, we compare our mercurial signature to related constructions. The closest work to ours is Mir et al. [50], and we highlight the differences. In particular, our construction supports thresholdization and DAC where [50] did not. Supporting DAC increases the size of public keys from [50].

1.2 Technical Overview

Our threshold mercurial signature takes the approach of Crites et al. [27] to thresholdize the secret key using Shamir secret sharing [56]. 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 [31].

Our scheme is based on the key-homomorphic signature construction in [50][3]. In this construction, a message[4] $(\vec{M}, \vec{N})$ with $\vec{M} = (M_1, M_2)$ and a tag[5] $\vec{T} =$

[3] [50] uses bilinear source groups $\mathbb{G}_1$ and $\mathbb{G}_2$ generated by P and $\hat{P}$ respectively

[4] The $\vec{N}$ vector is not important for explaining the key-homomorphic properties of the scheme, so we leave an explaination of this to later in our paper.

[5] Tags were added to support randomization properties. Intuitively, tags can be thought of as part of the message.

Schemes	[28]	[46]	[9]	[50]	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	$\times$	$\times$	✓	$\times^\dagger$	✓
Non-interactive	N/A‡	N/A‡	$\times$	✓	✓
Supports DAC	✓	✓	✓	$\times$	✓

Table 1. Concrete parameter comparison.

ℓ denotes the length of signable messages. "non-interactive" denotes issuers do not need to interact to each other for a signature to be combined. A non-constant $|\mathsf{pp}|$ (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 to 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 [50].

(T_1, T_2) are signed under a secret key $\mathsf{sk} = \{\mathsf{sk}_i\}_{i\in[5]}$ to produce a signature $\sigma = (h, b, s) = (h, T_1^{\mathsf{sk}_4} T_2^{\mathsf{sk}_5}, h^{\mathsf{sk}_1} M_1^{\mathsf{sk}_2} M_2^{\mathsf{sk}_3})$, where h is a hash of the tag and message. The latter two parts of the signature are key-homomorphic, since $b \cdot b' = T_1^{\mathsf{sk}_4+\mathsf{sk}'_4} T_2^{\mathsf{sk}_5+\mathsf{sk}'_5}$ and $s \cdot s' = h^{\mathsf{sk}_1+\mathsf{sk}'_1} M_1^{\mathsf{sk}_2+\mathsf{sk}'_2} M_2^{\mathsf{sk}_3+\mathsf{sk}'_3}$. So, as long as $h = h'$, $T_i = T'_i$, and $M_i = M'_i$, the product of two signatures is equivalent to a signature under the sum of the two secret keys, sk and sk'. We exploit this property to construct threshold signatures. A complete secret key $\mathsf{sk} = \{\mathsf{sk}_i\}_{i\in[5]}$ is shared using Shamir secret sharing into n shares $(\{\mathsf{sk}_{i,j}\}_{i\in[n],j\in[5]})$. For all $j \in [5]$ and $\mathfrak{T} \subset [n]$ satisfying $|\mathfrak{T}| = t$ for a threshold t, $\sum_{i\in\mathfrak{T}} \lambda_{i,\mathfrak{T}} \mathsf{sk}_{i,j} = \mathsf{sk}_j$, where $\lambda_{i,\mathfrak{T}}$ is the Lagrange coefficient for party i in $\mathfrak{T}$. By leveraging the signature's homomorphic properties, we can combine any set t of partial signatures into into a single signature that verifies under the original public key.

The intuition behind our second contribution (adapting [50] 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. Thus, signing a public key means extending it with extra elements in the first source group. In [50], a public key takes the form $\mathsf{pk} = \{\hat{P}^{\mathsf{sk}_i}\}_{i\in[5]}$ where $\hat{P}$ is the generator for the second source group. A naive approach for extending this key might try revealing $\mathsf{pk}' = \{P^{\mathsf{sk}_i}\}_{i\in[5]}$ (where P is the generator of the first source group), but this allows the computation of $\sigma^* = (h^* = P^{\mathsf{sk}_1}, b^* = P^{\mathsf{sk}_2} P^{\mathsf{sk}_3}, s^* = P^{\mathsf{sk}_4} P^{\mathsf{sk}_5})$, which is a forgery of the message $(\vec{M}^* = (P, P), \vec{N}^* = (\hat{P}, \hat{P}))$.

Fortunately, when these elements of the public key in the first source group are structured similar to messages and tags, we can prove the scheme to be unforgeable. By revealing: $\mathsf{pk}' = (T_1, \cdots, T_5, M_1, \cdots, M_5)$ where $T_i = h^{\rho_i}$ and $M_i = (T_i)^{\mathsf{sk}_i}$, we find that we can sign public keys using these additional elements. Furthermore, we prove in the generic group model that a forgery on this extended public key constitutes a forgery on the original scheme of [50]. This proof is highly non-trivial and requires that these extra elements can effectively be "simulated" by the reduction without knowing the secret key, which we prove in the GGM[6]. Finally, we generalize the scheme so that arbitrary length vectors ($\vec{M} = (M_1, \cdots, M_\ell)$) can be signed.

1.3 Related Work

Anonymous Credentials. Anonymous credentials (ACs) are finding more applications in recent years, like the EU's Digital Identity Wallet regulation (EU-DIW) [12], ISO's group signatures [1], TCG's Direct Anonymous Attestation protocols [2], and W3C's Decentralized Identifiers [3]. Several implementations and specifications have emerged, such as Hyperledger AnonCreds [4], Microsoft's uProve [54], and IBM's idemix [16]. Delegatable anonymous credentials (DACs) were first realized by Chase and Lysyanskaya [22] but were not made efficient for long chains until the work of Belenkiy et al. [13], which relied on Groth-Sahai proofs [47]. Since Groth-Sahai proofs are often considered inefficient, Crites and Lysyanskaya [28] recently introduced the notion of mercurial signatures as a more practical foundation for constructing efficient DACs.

Threshold Credentials. Threshold signatures have seen extensive use in blockchain technology due to their distributed nature [41,49,52,53,59]. A threshold signature scheme is parameterized by two integers, n and t, in addition to the security parameter. The signing key is divided into n shares, which are typically distributed among n parties that do not fully collude. During signing, t of these parties must come together with their shares and interact to produce a signature. Security requires that $n-t+1$ parties are honest. In this case, a malicious party controls at most $t-1$ shares, and thereby cannot forge a signature. Threshold signatures are also useful in multi-factor authentication [37], where the key used to show a signature is shared across multiple devices.

Mercurial Signatures. Mercurial signatures were introduced in [28] as a combination of equivalence classes on messages [40] and equivalence classes on public keys [10]. Mercurial signatures belong to a broader family of schemes with randomizable public keys, including *signatures with re-randomizable keys* [38], *key-blinded signatures*, [35] *signatures with honestly randomized keys*, [30] and *updatable signatures* [25]. For a disambiguation, see the work of Celi et al. [21]. Mercurial signatures were initially used to realize more efficient DACs that outperformed the previous construction built using Groth-Sahai proofs [13], but have also been used in other applications, such as contact tracing [44]. Some mercurial schemes [28, 50] suffer from a limitation: signature unlinkability only holds when the signer does not collude with the adversary. Griffy et al. [46] strengthened the definition and construction of mercurial signatures from [28]

[6] We describe this reduction in the proofs of unforgeability and hiding of Theorems 6 and 7.

to support privacy against malicious signers, though this construction requires a structured common reference string (SRS).

Many mercurial schemes are also *structure-preserving signatures* [5]. 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 [47]. These proofs are more efficient than black-box proofs because they allow algebraic relations between group elements to be proven directly without translating them into circuit representations of algebraic operations, as required in general-purpose ZKPs [43]). Moreover, Groth-Sahai proofs are *randomizable*, which is what allowed the efficient construction of DACs in [13].

Threshold Mercurial Signatures. Threshold mercurial signatures were first introduced by Abe et al. in [9], which adapted [28] to the threshold setting to create stronger anonymity and unforgeability properties for delegatable anonymous credentials. While their construction does not require an SRS (unlike [46]), their scheme does require interaction between signers to produce a signature, which is undesireable. Similar to threshold mercurial signatures is the the work of Mir et al. [50] which introduced *aggregatable* mercurial signatures. However, their scheme did not support the signing of public keys as messages [44], which limited their application to issuer-hiding [14] and multi-authority [48] ACs (which are implied by DAC, but not vice-versa). We note that when the threshold mercurial signatures set $t = 1$ with $n > t$ then the notion is closely related to group signatures [24].

1.4 Organization

Section 2 briefly presents our preliminaries and building blocks. Section 3 introduces our notion of threshold mercurial signature (TMS) and presents a construction with key size $\ell = 2$ that supports both thresholdization and the signing of public keys. Section 4 introduces the notion of threshold delegatable anonymous credentials (TDAC) along with a construction.

While the body of our paper is self-contained, we provide additional material in the appendix. When a section in the body relies on a definition from the appendix, we give intuition for the definition in the body. Appendix A details additional preliminaries and building blocks. Appendix B describes a TMS construction for an arbitrary key size ($\ell \in \mathsf{poly}(\lambda)$). Appendix C provides a detailed security proof of our TDAC construction. Appendices D and E prove the public key class-hiding and unforgeability, respectfully, of our TMS construction.

2 Preliminaries

In this section, we briefly introduce the notation, definitions, and building blocks that we use in the rest of the paper. Due to space constraints, we present the other preliminaries (i.e., mercurial signature, aggregated mercurial signature, etc.) alongside our construction in Section 3 and Appendix A.

Notation. We let $\lambda \in \mathbb{N}$ 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 $1 < t \leq n$. 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. A function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}$ is said to be negligible in λ if for every positive polynomial p, $\mathsf{negl}(\lambda) < 1/p(\lambda)$ when λ is sufficiently large. 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 denote by $x = \mathsf{val}$ or $x \leftarrow \mathsf{val}$ the assignment of a value val to the variable x. 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.

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 [56]. Shamir secret sharing allows a secret in a finite field (in this paper, $\mathbb{Z}_p$) to be split into n shares such that any subset of t shares can reconstruct the original. Any subset of fewer than t shares is uniformly distributed over $\mathbb{Z}_p$, making the scheme information-theoretically secure. In our construction, we will use the functions Share which splits a secret into n shares, and $\mathsf{Reconst}$, which recovers the secret from any t shares. To briefly review, Share will split an element x of $\mathbb{Z}_p$ into shares, $\{x_i\}_{i \in [n]}$, such that for any $\mathfrak{T} \subset [n]$, $\sum_{i \in \mathfrak{T}} \lambda_{\alpha, i, \mathfrak{T}} x_i = x$. We review this primitive more formally in Appendix A.1.

Tag-based Message Spaces. Following [27, 28, 50], 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 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 [47] and allowed for efficient DAC constructions [13]. However, it was later shown that using only a single source group leads to impossibility results concerning signatures size [6]. This limitation was mitigated by defining message spaces over multiple source groups [42] (*e.g.*, $(\vec{M}, \vec{N}) \in \mathbb{G}_1^\ell \times \mathbb{G}_2^\ell$), which are referred to as *Diffie-Hellman message spaces* [27].

To see why tags are important, recall the signatures from Section 1, which takes the form $\sigma = (h, b, s) = (h, T_1^{\mathsf{sk}_4} T_2^{\mathsf{sk}_5}, h^{\mathsf{sk}_1} M_1^{\mathsf{sk}_2} M_2^{\mathsf{sk}_3})$. One way to ensure unforgeability is to have the signer uniformly sample $h \xleftarrow{\$} \mathbb{G}_1$ for each signature. Randomly sampling h in this way prevents a forgery attack. If two distinct tag-message pairs signed with the same h, this yields $\sigma = (h, b, s)$ and $\sigma' = (h, b', s')$. A trivial forgery then follows by computing $b \cdot b' = (T_1 T_1')^{\mathsf{sk}_4} (T_2 T_2')^{\mathsf{sk}_5}$ and $s \cdot s' = h^{2\mathsf{sk}_1} (M_1 M_1')^{\mathsf{sk}_2} (M_2 M_2')^{\mathsf{sk}_3}$, producing a signature $\sigma^* = (h^2, bb', ss')$ for the message $\vec{M}^* = (M_1 M_1', M_2 M_2')$ and tag $\vec{T}^* = (T_1 T_1', T_2 T_2')$. Yet, to perform aggregation in [50] (and enable thresholdization in our signature), signatures must share the same first element h. This creates a problem, since we need equivalent messages to share h values, but we need to ensure that distinct messages do not share an h value to prevent forgeries. To resolve this, we let h be the output of a hash computed on both the tag and the message. This ensures, by collision resistance, that no two distinct tags and messages share an h value.

For more information on tags, see [50]. In our proofs, we do not use the internal structure of tags, but instead reduce directly to their construction. We describe the tag and message spaces in Definitions 1 and 2. The tag space is parameterized by part of the message vector, $\vec{N}$. Valid message/tag pairs must satisfy that $(\vec{M}, \vec{N}) \in \mathcal{M}$ and $\vec{T} \in \mathcal{T}^{\vec{N}}$. This resolves the "circular dependency" mentioned in [27][7]. This tag must be presented with the signature and message and thus to achieve meaningful anonymity properties, we must allow for randomization of tags, which can be done by randomizing the γ value in Definition 1.

Definition 1 (Tag Space ($\mathcal{T}^{\vec{N}}$) [50]). *$\mathcal{T}^{\vec{N}} \subset (\mathbb{G}_1)^\ell$ is a tag space if for all $\vec{T} = (T_1, \dots, T_\ell) \in \mathcal{T}^{\vec{N}}$, $\exists\, \gamma \in \mathbb{Z}_p^\times$, $\vec{\rho} = (\rho_1, \dots, \rho_\ell)$ such that $\vec{N} = (N_1, \dots, N_\ell) \in (\mathbb{G}_2)^\ell$, $h = H(P^{\rho_1} || \dots || P^{\rho_\ell} || N_1 || \dots || N_\ell)$ and $\vec{T} = (h^{\gamma\rho_1}, \dots, h^{\gamma\rho_\ell})$.*

Definition 2 (Tag-based DH Message Space ($\mathcal{M}$) [50]). *$\mathcal{M} \subset (\mathbb{G}_1)^\ell \times (\mathbb{G}_2)^\ell$ is a tag-based Diffie-Hellman (DH) message space if for all $(\vec{M}, \vec{N}) = (M_1, \dots, M_\ell, N_1, \dots, N_\ell) \in \mathcal{M}$, $\exists \vec{T} \in \mathcal{T}^{\vec{N}}$ such that and $i \in [\ell]$, $e(M_i, \hat{P}) = e(T_i, N_i)$.*

We omit the superscript $(\mathcal{T})$ to denote the union of tag spaces parameterized by all $\vec{N}$. Similar to [27, 50], since h depends on $\vec{T}$ and $\vec{M}$ depends on $(\vec{T}, h)$, to generate messages, we first sample $m_1, \dots, m_\ell, \rho_1, \dots, \rho_\ell$ from $\mathbb{Z}_p^\times$. Then, $\vec{N} = (\hat{P}^{m_1}, \dots, \hat{P}^{m_\ell})$ and $h = H(P^{\rho_1} || \dots || P^{\rho_\ell} || \hat{N}_1 || \dots || \hat{N}_\ell)$ are computed. $\vec{M}$ can be computed easily from there. This generation is described in the scheme by in the function $\mathsf{GenAuxTag}$ in Definition 3 with corresponding verification

[7] This was mentioned w.r.t. indexed message spaces, but the dependency was present in tag-based message spaces in [50]

function, VerifyAux[8]. Note that while signing messages and tags requires the secrets, $\rho_1, \ldots, \rho_\ell$ to be known to verify that the message and tags are valid, during verification, only the group elements, $\vec{T}, (\vec{M}, \vec{N})$, are required and thus the scheme is still structure preserving.

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.3. Threshold mercurial signatures imply non-threshold mercurial signatures as can be seen by setting $n = t = 1$.

3 Threshold Mercurial Signatures

In this section, we formally define the notion of a *non-interactive threshold mercurial signature* (TMS) which extends mercurial signatures with a distributed key-generation process and a threshold signing procedure, requiring a quorum of parties to jointly produce a signature. Unlike the construction in [9], our TMS scheme does not require any interaction between the signers during signature generation. To obtain a signature, a user needs to interact with t different signers independently via ParSign to receive t partial signatures, which they can then combine into one signature via Reconst.

3.1 Syntax and Security Definition

We first describe the syntax of threshold mercurial signatures in Definition 3 and its randomization properties: message class-hiding in Definition 6, public key class-hiding in Definition 8, and origin-hiding in Definition 9 as well as what it means to be unforgeable in Definition 5. We then describe the property necessary to build threshold delegatable anonymous credentials (TDAC) from this signature scheme (cross-scheme correctness) in Definition 12.

Intuition of randomization features. The main feature of mercurial signatures is their randomization properties. A signature holder can randomize the public key, message, and signature itself to still verify while being unlinkable to the original signature. Specifically, mercurial signature schemes provide a ChangeRep function that takes a verifying message and signature pair and outputs a randomized pair that still verifies. Similarly, these schemes provide a ConvertPK function to randomize a public key, and a ConvertSig function to update associated signatures accordingly. Since our definition is for a threshold mercurial signature, it modifies some of these functions to operate on partial keys.

[8] The "aux" part of VerifyAux is an artifact from the aggregation property of [50] which we inherit for consistency. $\mathsf{aux} = \perp$ for our construction.

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 introducting *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 $[(\vec{M}, \vec{N})]_{RelM})$. Unforgeability (Definition 5) can then be defined to require that the forged message doesn't belong to the equivalence class of any message queried to the signing oracle.

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 $[\vec{T}]_{\mathcal{R}_\mathcal{T}} = \{\vec{T}' : \exists \gamma \in \mathbb{Z}_p^\times, \vec{T}' = \vec{T}^\gamma\}$. We also want our unforgeability definition to capture forgeries when the public key is randomized. Thus, our unforgeability definition must also allow for the adversary to try to forge under any key that is in the same equivalence class as the challenge key and we define an equivalence relation for keys. For our construction without cross-scheme correctness, this equivalence class is $[\vec{N}]_{\mathcal{R}_\mathcal{K}} = \{\vec{N}' : \exists \nu \in \mathbb{Z}_p^\times, \vec{N}' = \vec{N}^\nu\}$. When we add cross-scheme correctness, the equivalence class of keys becomes identical to the equivalence class for messages.

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 of that message (*i.e.*, $(\vec{M}', \vec{N}') \xleftarrow{\$} [(\vec{M}, \vec{N})]$) from a new, freshly sampled message (*i.e.*, $(\vec{M}', \vec{N}') \xleftarrow{\$} \mathcal{M}$). Public key class-hiding is similar in that it ensures that it is difficult to distinguish a public key and its randomization from two independently sampled public keys. Furthermore, in the threshold public key class-hiding game, the adversary is given access to a partial signature oracle for each of the public keys. Finally, origin hiding (Definition 9) ensures that the scheme's randomization algorithms all have uniformly distributed outputs, so that they appear identical to a fresh sampling.

We now proceed with the syntax formal definitions of these properties of mercurial signatures. In our $\mathsf{ConvertTag}$, $\mathsf{ConvertSK}$, $\mathsf{ConvertPK}$, $\mathsf{ConvertSig}$, and $\mathsf{ChangeRep}$ functions, the converter is sampled uniformly from $\mathbb{Z}_p^\times$ if omitted. To facilitate the verification of tags, the functions $\mathsf{GenAuxTag}$, $\mathsf{VerifyTag}$, and $\mathsf{VerifyAux}$ are included. We described how $\mathsf{GenAuxTag}$ and $\mathsf{VerifyAux}$ are used to verify tags in Section 2. $\mathsf{VerifyTag}$ is only used in the security games.

Definition 3 (Threshold Mercurial Signatures). *A threshold mercurial signature scheme* TMS *is parameterized by a tag, message, and key space,* $\mathcal{T}$*,* $\mathcal{M}$*, and* $\mathcal{K}$*; equivalence relations for tags, messages, and keys,* $[\cdot]_{\mathcal{R}_\mathcal{T}}$*,* $[\cdot]_{\mathcal{R}_\mathcal{M}}$*, and* $[\cdot]_{\mathcal{R}_\mathcal{K}}$*; and consists of a tuple of the following* PPT *algorithms* (Setup*,*

KeyGen, $\mathsf{GenAuxTag}$, $\mathsf{VerifyAux}$, $\mathsf{VerifyTag}$, $\mathsf{ParSign}$, $\mathsf{ParVerify}$, $\mathsf{Reconst}$, Verify, $\mathsf{ConvertTag}$, $\mathsf{ConvertParSK}$, $\mathsf{ConvertParPK}$, $\mathsf{ConvertPK}$, $\mathsf{ConvertSig}$, $\mathsf{ChangeRep}$), *which have following syntax:*

- $\mathsf{Setup}(1^\lambda, 1^\ell) \to \mathsf{pp}$*: The setup algorithm takes as input the security parameter* λ *and key size* ℓ*, and outputs the public parameters* pp*. We assume that* pp *is known to the rest of the algorithms, even if omitted.*
- $\mathsf{KeyGen}(\mathsf{pp}, n, t) \to (\vec{\mathsf{sk}}, \vec{\mathsf{pk}}, \mathsf{pk}_0)$*: The key-generation algorithm takes as input the public parameters pp and integers* $t, n \in \mathsf{poly}(\lambda)$ *such that* $1 \le t \le n$*, and outputs the vectors of partial signing and public keys* $\vec{\mathsf{sk}} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ *and* $\vec{\mathsf{pk}} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$*, and the global public key* pk_0*. Each party* P_i *for* $i \in [n]$ *receives* pk_0 *and their key pair* $(\mathsf{sk}_i, \mathsf{pk}_i)$*.*
- $\mathsf{GenAuxTag}(\{m_1, \ldots, m_\ell\}) \to (\vec{\rho}, \vec{T}, (\vec{M}, \vec{N}), \mathsf{aux})$*: The auxiliary tag generation algorithm takes as input a message secret* $\{m_1, \ldots, m_\ell\}$ *and outputs a tag secret* $\vec{\rho}$*, a public tag* $\vec{T}$*, a message such that* $(\vec{M}, \vec{N}) \in \mathcal{M}^{\vec{T}}$ *as described in Section 2, and auxiliary data* aux.
- $\mathsf{VerifyAux}(\mathsf{aux}, \vec{\rho}, (\vec{M}, \vec{N})) \to \{0, 1\}$*: The auxiliary verification algorithm takes as input an auxiliary data* aux*, tag secret* $\vec{\rho}$*, and message vectors* $(\vec{M}, \vec{N}) \in \mathcal{M}$*, and outputs* 1 *if they are correctly distributed and* 0 *otherwise.*
- $\mathsf{VerifyTag}(\vec{\rho}, \vec{T}) \to \{0, 1\}$*: The tag verification algorithm takes as input a tag* $\vec{T}$ *and its secret* $\vec{\rho}$*, and outputs* 1 *if they are valid (i.e.,* $\vec{T} \in \mathcal{T}$*) and* 0 *otherwise.*
- $\mathsf{ParSign}(\mathsf{sk}_i, \vec{\rho}, \mathsf{aux}, (\vec{M}, \vec{N})) \to \sigma_i$*: The partial signature algorithm takes as input a partial signing key* sk_i*, tag secret* $\vec{\rho}$*, auxiliary data* aux*, and message vectors* $(\vec{M}, \vec{N}) \in \mathcal{M}$ *and outputs a partial signature* σ_i*.*
- $\mathsf{ParVerify}(\mathsf{pk}_i, \vec{T}, (\vec{M}, \vec{N}), \sigma_i) \to \{0, 1\}$*: The partial signature verification algorithm takes as input a partial public key* pk_i*, public tag* $\vec{T}$*, message vectors* $(\vec{M}, \vec{N}) \in \mathcal{M}$*, and partial signature* σ_i*, and outputs* 1 *if* σ_i *is a valid signature and* 0 *otherwise.*
- $\mathsf{Reconst}(\vec{\mathsf{pk}}, \vec{T}, (\vec{M}, \vec{N}), \{i, \sigma_i\}_{i \in \mathfrak{T}}) \to \sigma$*: The signature reconstruction algorithm takes as input the vector of all partial public keys* $\vec{\mathsf{pk}}$*, message vectors* $(\vec{M}, \vec{N}) \in \mathcal{M}$*, and a set of* $\mathfrak{T} \subset [n]$ *(such that* $|\mathfrak{T}| = t$*) partial signatures* σ_i *with corresponding indices* i*, and outputs a combined signature* σ*.*
- $\mathsf{Verify}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma) \to \{0, 1\}$*: The signature verification algorithm takes as input the global public key* pk_0*, public tag* $\vec{T}$*, message vectors* $(\vec{M}, \vec{N}) \in \mathcal{M}$*, and signature* σ *and outputs* 1 *if* σ *is a valid signature or* 0 *otherwise.*
- $\mathsf{ConvertTag}(\vec{T}; \gamma) \to \vec{T}'$*: The tag conversion algorithm takes as input a tag* $\vec{T}$ *and converter* $\gamma \in \mathbb{Z}_p^\times$*, and outputs a new tag* $\vec{T}' \in [\vec{T}]_{\mathcal{R}_\mathcal{T}}$*.*
- $\mathsf{ConvertParSK}(\mathsf{sk}_i; \omega) \to \mathsf{sk}'_i$*: The secret key conversion algorithm takes as input a partial signing key* sk_i *and converter* $\omega \in \mathbb{Z}_p^\times$*, and outputs a new signing key* $\mathsf{sk}'_i \in [\mathsf{sk}_i]_{\mathcal{R}_{\mathcal{SK}}}$*.*
- $\mathsf{ConvertParPK}(\mathsf{pk}_i; \omega) \to \mathsf{pk}'_i$*: The public key conversion algorithm takes as input a partial public key* pk_i *and converter* $\omega \in \mathbb{Z}_p^\times$*, and outputs a new public key* $\mathsf{pk}'_i \in [\mathsf{pk}_i]_{\mathcal{R}_\mathcal{K}}$*.*

– $\mathsf{ConvertPK}(\mathsf{pk}_0;\omega) \to \mathsf{pk}_0'$: *The public key conversion algorithm takes as input the global public key* pk_i *and converter* $\omega \in \mathbb{Z}_p^\times$*, and outputs a new public key* $\mathsf{pk}_0' \in [\mathsf{pk}_0]_{\mathcal{R}_\mathcal{K}}$.

– $\mathsf{ConvertSig}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma; \omega) \to \sigma'$: *The signature conversion algorithm takes as input the global public key* pk_0*, public tag* $\vec{T}$*, message vectors* $(\vec{M}, \vec{N}) \in \mathcal{M}$*, signature* σ *and converter* $\omega \in \mathbb{Z}_p^\times$*, and outputs a new signature* σ' *that verifies with* $\mathsf{pk}_0' = \mathsf{ConvertPK}(\mathsf{pk}_0;\omega)$

– $\mathsf{ChangeRep}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma; (\mu,\nu)) \to (\vec{T}', (\vec{M}', \vec{N}'), \sigma')$: *On input the global public key* pk_0*, public tag* $\vec{T}$*, message vectors* $(\vec{M}, \vec{N}) \in \mathcal{M}$*, signature* σ*, and converters* $\mu, \nu \in \mathbb{Z}_p^\times$ *and outputs a new tag* $\vec{T}' \in [\vec{T}]_{\mathcal{R}_\mathcal{T}}$*, message* $(\vec{M}', \vec{N}') \in [(\vec{M}, \vec{N})]_{\mathcal{R}_\mathcal{M}}$*, and signature* σ'.

A TMS *scheme must satisfy the properties of correctness (Definition 4), unforgeability (Definition 5), message class-hiding (Definition 6), tag class hiding (Definition 7), public key class-hiding (Definition 8), and origin-hiding (Definition 9).*

Definition 4 (Correctness). *A threshold mercurial signature scheme* TMS = (Setup*,* KeyGen*,* GenAuxTag*,* VerifyAux*,* VerifyTag*,* ParSign*,* ParVerify*,* Reconst*,* Verify*,* ConvertTag*,* ConvertParSK*,* ConvertParPK*,* ConvertPK*,* ConvertSig*,* ChangeRep) *is correct if, for all* $\lambda, n, t \in \mathbb{N}$ *such that* $1 \le t \le n$*, for all* $(\vec{M}, \vec{N}) \in \mathcal{M}$*, for all* $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$*, for all* $(\vec{\mathsf{sk}}, \vec{\mathsf{pk}}, \mathsf{pk}_0) \in \mathsf{KeyGen}(\mathsf{pp}, n, t)$*, and for all* $\vec{M}, \vec{N}, \vec{\rho}$ *and* $\vec{T} \in \mathsf{GenAuxTag}(\vec{M}, \vec{N})$ *such that* $\mathsf{VerifyTag}(\vec{\rho}, \vec{T}) = 1$*, it satisfies the following conditions:*

– *Partial Verification. For all* $\sigma_i \in \mathsf{ParSign}(\mathsf{sk}_i, \vec{\rho}, \mathsf{aux}, (\vec{M}, \vec{N}))$*, it holds that* $\mathsf{ParVerify}(\mathsf{pk}_i, \vec{T}, (\vec{M}, \vec{N}), \sigma_i) = 1$.

– *Threshold Verification. For all* $\mathfrak{T} \subseteq [n]$ *of size* $|\mathfrak{T}| = t$ *and for all* $\sigma \leftarrow \mathsf{Reconst}(\vec{\mathsf{pk}}, \vec{T}, (\vec{M}, \vec{N}), \{i, \mathsf{ParSign}(\mathsf{sk}_i, \vec{\rho}, \mathsf{aux}, (\vec{M}, \vec{N}))\}_{i \in \mathfrak{T}})$*, it holds that* $\mathsf{Verify}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma) = 1$.

– *Key Conversion. For all* $(\vec{\mathsf{sk}}, \vec{\mathsf{pk}}, \mathsf{pk}_0) \in \mathsf{KeyGen}(\mathsf{pp}, n, t)$*,* $\vec{\mathsf{sk}}' \in (\mathsf{ConvertParSK}(\mathsf{sk}_i))_{i \in [n]}$*,* $\vec{\mathsf{pk}} \in (\mathsf{ConvertParPK}(\mathsf{pk}_i))_{i \in [n]}$*, and* $\mathsf{pk}_0' \in \mathsf{ConvertPK}(\mathsf{pk}_0)$*, it holds that* $(\vec{\mathsf{sk}}', \vec{\mathsf{pk}}', \mathsf{pk}_0') \in \mathsf{KeyGen}(\mathsf{pp}, n, t)$.

– *Signature Conversion. For all* σ *s.t.* $\mathsf{Verify}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma) = 1$ *and for all* $\sigma' \in \mathsf{ConvertSig}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma)$*, it holds that* $\mathsf{Verify}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma') = 1$.

– *Change of Message Representative. For all* σ *s.t.* $\mathsf{Verify}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma) = 1$ *and for all* $(\vec{T}', (\vec{M}', \vec{N}'), \sigma') \in \mathsf{ChangeRep}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma))$*, it holds that* $\mathsf{Verify}(\mathsf{pk}_0, \vec{T}', (\vec{M}', \vec{N}'), \sigma') = 1$ *and* $(\vec{M}', \vec{N}') \in [(\vec{M}, \vec{N})]_{\mathcal{R}_\mathcal{M}}$.

Definition 5 (Threshold Unforgeability). *A threshold mercurial signature scheme* TMS = (Setup*,* KeyGen*,* GenAuxTag*,* VerifyAux*,* VerifyTag*,* ParSign*,* ParVerify*,* Reconst*,* Verify*,* ConvertTag*,* ConvertParSK*,* ConvertParPK*,* ConvertPK*,* ConvertSig*,* ChangeRep) *is existentially unforgeable under adaptively chosen* *tagged message attack* (EUF-CtMA) *if, for all* $\lambda, n, t \in \mathbb{N}$ *s.t.* $1 \leq t \leq n$, *and for all* PPT *adversaries* $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CtMA}}_{\mathsf{TMS},\mathcal{A}} = \Pr\left[\mathsf{EUF\text{-}CtMA}^{\mathsf{TMS}}_{\mathcal{A}}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

where the unforgeability game $\mathsf{EUF\text{-}CtMA}^{\mathsf{TMS}}_{\mathcal{A}}(1^\lambda)$ *is defined in Figure 1.*

```
$\mathsf{EUF\text{-}CtMA}^{\mathsf{TMS}}_{\mathcal{A}}(1^\lambda)$
Initialize $Q \leftarrow \emptyset$; $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\vec{\mathsf{sk}}, \vec{\mathsf{pk}}, \mathsf{pk}_0) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, n, t)$
$(\mathcal{C}, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{pk}_0, \vec{\mathsf{pk}})$
$(\vec{T}^*, \vec{\rho}^*, (\vec{M}^*, \vec{N}^*), \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{PSign}}}(\mathsf{st}, \{\mathsf{sk}_i\}_{i\in\mathcal{C}}, \vec{\mathsf{pk}}, \mathsf{pk}_0)$
if $\mathsf{Verify}(\mathsf{pk}_0, \vec{T}^*, (\vec{M}^*, \vec{N}^*), \sigma^*) = 1 \wedge \mathcal{C} \in [n] \wedge |\mathcal{C}| < t$
    $\wedge\ \forall(\vec{M}, \vec{N}) \in Q, [(\vec{M}^*, \vec{N}^*)]_{\mathcal{R}_\mathcal{M}} \neq [(\vec{M}, \vec{N})]_{\mathcal{R}_\mathcal{M}}$
    $\wedge\ \mathsf{VerifyTag}(\vec{\rho}, \vec{T}) = 1$
  return 1
else return 0

$\mathcal{O}^{\mathsf{PSign}}(i, \vec{\rho}, \mathsf{aux}, (\vec{M}, \vec{N}))$
if $i \notin [n]$ then
  return $\perp$
else
  $\sigma_i \leftarrow \mathsf{ParSign}(\mathsf{sk}_i, \vec{\rho}, \mathsf{aux}, (\vec{M}, \vec{N}))$
  $Q \leftarrow Q \cup (\vec{M}, \vec{N})$
  return $\sigma_i$
```

Fig. 1. Unforgeability Game $\mathsf{EUF\text{-}CtMA}^{\mathsf{TMS}}_{\mathcal{A}}(1^\lambda)$ for TMS scheme.

Definition 6 (Message Class-Hiding). *A threshold mercurial signature scheme* TMS = (Setup, KeyGen, GenAuxTag, VerifyAux, VerifyTag, ParSign, ParVerify, Reconst, Verify, ConvertTag, ConvertParSK, ConvertParPK, ConvertPK, ConvertSig, ChangeRep) *is message class-hiding if, for all* $\lambda, n, t \in \mathbb{N}$ *such that* $1 \leq t \leq n$, *and for all* PPT *adversaries* $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{MSG\text{-}CLH}}_{\mathsf{TMS},\mathcal{A}} = \Pr\left[\mathsf{MSG\text{-}CLH}^{\mathsf{TMS}}_{\mathcal{A}}(1^\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where the message class-hiding game $\mathsf{MSG\text{-}CLH}^{\mathsf{TMS}}_{\mathcal{A}}(1^\lambda)$ *is defined in Figure 2.*

Definition 7 (Tag Class-Hiding). *A threshold mercurial signature scheme* TMS = (Setup, KeyGen, GenAuxTag, VerifyAux, VerifyTag, ParSign, ParVerify, Reconst, Verify, ConvertTag, ConvertParSK, ConvertParPK, ConvertPK, ConvertSig, ChangeRep) *is tag class-hiding if, for all* $\lambda, n, t \in \mathbb{N}$ *such that* $1 \leq t \leq n$, *and for all* PPT *adversaries* $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{TAG\text{-}CLH}}_{\mathsf{TMS},\mathcal{A}} = \Pr\left[\mathsf{TAG\text{-}CLH}^{\mathsf{TMS}}_{\mathcal{A}}(1^\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where the tag class-hiding game $\mathsf{TAG\text{-}CLH}^{\mathsf{TMS}}_{\mathcal{A}}(1^\lambda)$ *is defined in Figure 2.*

```
$\mathsf{MSG\text{-}CLH}^{\mathsf{TMS}}_{\mathcal{A}}(1^\lambda)$

1 :  $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
2 :  $(\vec{M}_1, \vec{N}_1) \overset{\$}{\leftarrow} \mathcal{M}$
3 :  $(\vec{M}^0_2, \vec{N}^0_2) \overset{\$}{\leftarrow} \mathcal{M}$
4 :  $(\vec{M}^1_2, \vec{N}^1_2) \overset{\$}{\leftarrow} [(\vec{M}_1, \vec{N}_1)]_{\mathcal{R}_\mathcal{M}}$
5 :  $b \overset{\$}{\leftarrow} \{0,1\}$
6 :  $b' \leftarrow \mathcal{A}(\mathsf{pp}, (\vec{M}_1, \vec{N}_1), (\vec{M}^b_2, \vec{N}^b_2))$
7 :  return $b = b'$

$\mathsf{TAG\text{-}CLH}^{\mathsf{TMS}}_{\mathcal{A}}(1^\lambda)$

1 :  $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
2 :  $\vec{T}_1 \overset{\$}{\leftarrow} \mathcal{T}$
3 :  $\vec{T}^0_2 \overset{\$}{\leftarrow} \mathcal{T}$
4 :  $\vec{T}^1_2 \overset{\$}{\leftarrow} [\vec{T}_1]_{\mathcal{R}_\mathcal{T}}$
5 :  $b \overset{\$}{\leftarrow} \{0,1\}$
6 :  $b' \leftarrow \mathcal{A}(\mathsf{pp}, \vec{T}_1, \vec{T}^b_2)$
7 :  return $b = b'$
```

```
$\mathsf{PK\text{-}CLH}^{\mathsf{TMS}}_{\mathcal{A}}(1^\lambda)$

1 :  $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
2 :  $(\vec{\mathsf{sk}}_1, \vec{\mathsf{pk}}_1, \mathsf{pk}_{1,0}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, n, t)$
3 :  $(\mathcal{C}, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{pk}_{1,0}, \vec{\mathsf{pk}}_1)$
4 :  $(\vec{\mathsf{sk}}^0_2, \vec{\mathsf{pk}}^0_2, \mathsf{pk}^0_{2,0}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, n, t)$
5 :  $\omega \overset{\$}{\leftarrow} \mathbb{Z}^\times_p$
6 :  for $i \in [n]$ do
7 :    $\mathsf{sk}^1_{2,i} \leftarrow \mathsf{ConvertSK}(\mathsf{sk}_{1,i}, \omega)$
8 :  $\vec{\mathsf{pk}}^1_2 \leftarrow \mathsf{ConvertPK}(\vec{\mathsf{pk}}_1, \omega)$
9 :  $b \overset{\$}{\leftarrow} \{0,1\}$
10 : $b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{PSign}}}(\mathsf{st}, \{\mathsf{sk}_{1,i}, \mathsf{sk}^b_{2,i}\}_{i\in\mathcal{C}}, \mathsf{pk}^b_2, \vec{\mathsf{pk}}^b_2)$
11 : return $b = b'$

$\mathcal{O}^{\mathsf{PSign}}(i, \vec{\rho}, \mathsf{aux}, (\vec{M}, \vec{N}))$

1 :  if $i \notin [n]$ then
2 :    return $\perp$
3 :  else
4 :    $\sigma_{1,i} \leftarrow \mathsf{ParSign}(\mathsf{sk}_{1,i}, \vec{\rho}, \mathsf{aux}, (\vec{M}, \vec{N}))$
5 :    $\sigma_{2,i} \leftarrow \mathsf{ParSign}(\mathsf{sk}^b_{2,i}, \vec{\rho}, \mathsf{aux}, (\vec{M}, \vec{N}))$
6 :    return $(\sigma_{1,i}, \sigma_{2,i})$
```

Fig. 2. Message class-hiding game $\mathsf{MSG\text{-}CLH}^{\mathsf{TMS}}_{\mathcal{A}}(1^\lambda)$, tag class-hiding game $\mathsf{TAG\text{-}CLH}^{\mathsf{TMS}}_{\mathcal{A}}(1^\lambda)$, public key class-hiding game $\mathsf{PK\text{-}CLH}^{\mathsf{TMS}}_{\mathcal{A}}(1^\lambda)$ for TMS scheme.

Definition 8 (Threshold Public Key Class-Hiding). *A threshold mercurial signature scheme* TMS = (Setup, KeyGen, $\mathsf{GenAuxTag}$, $\mathsf{VerifyAux}$, $\mathsf{VerifyTag}$, $\mathsf{ParSign}$, $\mathsf{ParVerify}$, $\mathsf{Reconst}$, Verify, $\mathsf{ConvertTag}$, $\mathsf{ConvertParSK}$, $\mathsf{ConvertParPK}$, $\mathsf{ConvertPK}$, $\mathsf{ConvertSig}$, $\mathsf{ChangeRep}$) *is public key class-hiding if, for all* $\lambda, n, t \in \mathbb{N}$ *such that* $1 \le t \le n$*, and for all* PPT *adversaries* $\mathcal{A}$*,*

$$\mathsf{Adv}^{\mathsf{PK\text{-}CLH}}_{\mathsf{TMS},\mathcal{A}} = \Pr\left[\mathsf{PK\text{-}CLH}^{\mathsf{TMS}}_{\mathcal{A}}(1^\lambda) = 1\right] \le \frac{1}{2} + \mathsf{negl}(\lambda)$$

where the game $\mathsf{PK\text{-}CLH}^{\mathsf{TMS}}_{\mathcal{A}}(1^\lambda)$ *is defined in Figure 2.*

Definition 9 (Origin-Hiding). *A threshold mercurial signature scheme* (Setup, KeyGen, $\mathsf{GenAuxTag}$, $\mathsf{VerifyAux}$, $\mathsf{VerifyTag}$, $\mathsf{ParSign}$, $\mathsf{ParVerify}$, $\mathsf{Reconst}$, Verify, $\mathsf{ConvertTag}$, $\mathsf{ConvertParSK}$, $\mathsf{ConvertParPK}$, $\mathsf{ConvertPK}$, $\mathsf{ConvertSig}$, $\mathsf{ChangeRep}$) *is origin-hiding if, for all* $\lambda, n, t \in \mathbb{N}$ *such that* $1 \le t \le n$*, for all* $(\vec{M}, \vec{N}) \in \mathcal{M}$*, for all* $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$*, for all* $(\vec{\mathsf{sk}}, \vec{\mathsf{pk}}, \mathsf{pk}_0) \in \mathsf{KeyGen}(\mathsf{pp}, n, t)$*, for all* aux *and* $\vec{\rho}$ *such that* $\mathsf{VerifyAux}(\mathsf{aux}, \vec{\rho}, (\vec{M}, \vec{N})) = 1$*, and for* $\sigma \leftarrow \mathsf{Reconst}(\vec{\mathsf{pk}}, \vec{T}, (\vec{M}, \vec{N}), \{i, \sigma_i\}_{i\in\mathfrak{T}}))$ *for some set* $\mathfrak{T} \subset [n]$ *of size* $|\mathfrak{T}| = t$*, it satisfies the following properties:*

– ***Origin-Hiding of*** $\mathsf{ConvertPK}$***:*** *For all* $\omega \in \mathbb{Z}_p^\times$*,* $\mathsf{pk}' = \mathsf{ConvertPK}(\mathsf{pk}; \omega)$ *is a uniform random element in* $[\mathsf{pk}]_{\mathcal{R}_\mathcal{K}}$*.*
– ***Origin-Hiding of*** $\mathsf{ConvertSig}$***:*** *For all* $\omega \in \mathbb{Z}_p^\times$*,* sk_0*,* $(\vec{M}, \vec{N})$*, and* $\sigma \in \mathsf{Sign}(\mathsf{sk}_0, (\vec{M}, \vec{N}))$*,* $\sigma' = \mathsf{ConvertSig}(\sigma; \omega)$ *is a uniform random element in the image of* $\mathsf{Sign}(\mathsf{sk}_0, (\vec{M}, \vec{N}))$
– ***Origin-Hiding of*** $\mathsf{ChangeRep}$***:*** *For all* $(\mu, \nu) \in (\mathbb{Z}_p^\times)^2$*,* $(\vec{T}', (\vec{M}', \vec{N}'), \sigma') \leftarrow \mathsf{ChangeRep}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma; (\mu, \nu))$ $\vec{T}'$ *is uniform in* $[\vec{T}]_{\mathcal{R}_\mathcal{T}}$ *and* $(\vec{M}', \vec{N}')$ *is uniform in* $[(\vec{M}, \vec{N})]_{\mathcal{R}_\mathcal{M}}$*.*
– ***Origin-Hiding of*** $\mathsf{ConvertTag}$***:*** *For all* $\gamma \in \mathbb{Z}_p^\times$*,* $\vec{T}' \leftarrow \mathsf{ConvertTag}(\vec{T}; \gamma)$*,* $\vec{T}'$ *is uniform in* $[\vec{T}]_{\mathcal{R}_\mathcal{T}}$*.*

For brevity, we informally define unforgeability and public key class-hiding of non-thresholdized mercurial signatures w.r.t. our thresholdized definitions in Definitions 10 and 11. Intuitively, we can simply set $n = t = 1$ and redefine $\mathsf{ParSign}$ in the PKCH and unforgeability games to obtain correct definitions for non-thresholdized schemes. We review the formal definitions of non-thresholdized mercurial signatures in Appendices A.3 and A.4. The non-threshold versions of tag and message class-hiding are identical.

Definition 10 (Unforgeability (informal)). *A mercurial signature scheme* MS *is existentially unforgeable under adaptively chosen tagged message attack* ($\mathsf{EUF\text{-}CtMA}$) *if for all* $\lambda \in \mathbb{N}$ *and* $t = n = 1$*, for all* PPT *adversaries* $\mathcal{A}$*,*

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CtMA}}_{\mathsf{MS},\mathcal{A}} = \Pr\left[\mathsf{EUF\text{-}CtMA}^{\mathsf{MS}}_{\mathcal{A}}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

where the unforgeability game $\mathsf{EUF\text{-}CtMA}^{\mathsf{MS}}_{\mathcal{A}}(1^\lambda)$ *is defined in Figure 1 with one modification:* Sign *is used in place of* $\mathsf{ParSign}$*.*

Definition 11 (Public Key Class-Hiding (informal)). *A mercurial signature scheme* MS *has public key class-hiding if for all* $\lambda \in \mathbb{N}$ *and* $t = n = 1$*, for all* PPT *adversaries* $\mathcal{A}$*,*

$$\mathsf{Adv}^{\mathsf{PK\text{-}CLH}}_{\mathsf{MS},\mathcal{A}} = \Pr\left[\mathsf{PK\text{-}CLH}^{\mathsf{MS}}_{\mathcal{A}}(1^\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where the public key class-hiding game $\mathsf{PK\text{-}CLH}^{\mathsf{MS}}_{\mathcal{A}}(1^\lambda)$ *is defined in Figure 2 with one modification:* Sign *is used in place of* $\mathsf{ParSign}$*.*

Cross-scheme correctness Some definitions of mercurial signatures, such as the one in [50], do not support the signing of public keys. This was an informal property of [28] that was later formalized as *cross-scheme correctness* in [44]. Cross-scheme correctness can be seen as a weaker version of *automorphic signatures* [7, 39] which requires that a signature scheme's public key space lies in the message space, allowing for public keys to be signed. While cross-scheme correctness doesn't require the public key space of the scheme to lie in the message space, it requires that the scheme can be extended into a new scheme such that the public key space lies in the message space of the extended scheme. In this paper, we define a property (similar to [44]) of our scheme called *leveled* cross scheme correctness (in Definition 12) that introduces the function $\mathsf{ExtendSetup}(\mathsf{pp}) \to \mathsf{pp}'$, which takes in public parameters and extends the scheme to a lower level for DAC. This ensures that keys can be signed, thus allowing for the (relatively) generic construction of DAC from mercurial signatures. $\mathsf{ExtendSetup}$ takes in the parameters for a mercurial signature scheme, pp, and creates a new scheme defined by a new set of parameters, pp', such that any public key generated by pp can be signed by pp'. Moreover, our public key generation function must output an extra "secret tag" value $\vec{\rho}$, which is used to sign public keys (similar to the $\vec{\rho}$ output by $\mathsf{GenAuxTag}$) which we include in each sk_i output by KeyGen.

Definition 12 (Leveled cross-scheme correctness). *A threshold mercurial signature scheme* TMS *is leveled cross-scheme correct if, for all* $n, t, \ell \in O(\lambda)$, $\mathsf{pp} \in \mathsf{Setup}(1^\lambda, 1^\ell)$*,* $\mathsf{pp}' \leftarrow \mathsf{ExtendSetup}(\mathsf{pp})$*,* $(\vec{\mathsf{sk}}, \vec{\mathsf{pk}}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, n, t)$*,* $(\vec{\mathsf{sk}}', \vec{\mathsf{pk}}', \mathsf{pk}') \leftarrow \mathsf{KeyGen}(\mathsf{pp}', n, t)$*,* $\forall i \in [n], \sigma_i \leftarrow \mathsf{Sign}(\mathsf{pp}', \mathsf{sk}'_i, \vec{\rho}, \mathsf{pk})$*,* $\forall \mathfrak{T} \subset [n]$*,* $|\mathfrak{T}| = t$*,* $\sigma = \mathsf{Reconst}(\vec{\mathsf{pk}}', \vec{\mathsf{pk}}, \{\sigma_i\}_{i \in \mathfrak{T}})$*, it holds that* $\mathsf{Verify}(\mathsf{pp}', \mathsf{pk}', \mathsf{pk}, \sigma) = 1$.

3.2 Enhanced Construction (adapted from [50])

Before explaining our threshold construction, we present the mercurial signature construction from [50] that our scheme is based on. Although the original scheme in [50] was an *aggregatable* mercurial signature, we remove the aggregatable features here by omitting $\mathsf{AggrSign}$ and $\mathsf{VerifyAggr}$ (since they aren't relevant to our paper), turning it into a regular mercurial signature. We also make some modifications to the notation to keep it consistent with construction. Finally, we modify the original scheme for cross-scheme correctness, including extra terms in $\boxed{\text{solid boxes}}$. We refer to the version of the scheme without these boxed elements as the plain version, and the version with these boxed elements as the cross-scheme correct version.

The equivalence classes for messages, tags, and public keys used in the plain version of the scheme are defined identically to [50]. To support cross-scheme correctness, we also add an equivalence class for public keys with cross-scheme correctness. We described these equivalence classes informally in Section 3.1. We formally review them all—including the one we added for cross-scheme correctness—in Appendix B.

This construction is presented in Figure 3 for the case where messages are of length $\ell = 2$ (*i.e.*, $(\vec{M}, \vec{N}) = (M_1, M_2, \hat{N}_1, \hat{N}_2)$ and $\vec{T} = (T_1, T_2)$). The generalized construction for any $\ell \in \mathsf{poly}(\lambda)$ can be found in Appendix B.1 by setting $n = t = 1$ in our generalized threshold mercurial signature construction. We refer to the construction in Figure 3 as $\mathbf{\Pi}_{\mathsf{MBGLS}}$, named after the authors of [50].

Theorem 1 (Correctness). *The* $\mathbf{\Pi}_{\mathsf{MBGLS}}$ *construction in Figure 3 is a correct mercurial signature scheme scheme as per Definition 4.*

```
$\mathsf{Setup}(1^\lambda)$
1 : $\mathsf{BG} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, \hat{P}, e) \leftarrow \mathsf{BGGen}(1^\lambda)$
2 : Sample a hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{G}_1$
3 : return $\mathsf{pp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, \hat{P}, e, \mathsf{H})$

$\mathsf{KeyGen}(\mathsf{pp})$
1 : Sample $\mathsf{sk} := (x, y_1, y_2, z_1, z_2) \xleftarrow{\$} \left(\mathbb{Z}_p^\times\right)^5$
2 : Sample $\boxed{(\rho_i)_{i \in [5]}} \xleftarrow{\$} \left(\mathbb{Z}_p^\times\right)^5$
3 : Compute $\mathsf{pk} = \left(\vec{N}^{(\mathsf{pk})}, \boxed{\vec{M}^{(\mathsf{pk})}}, \boxed{\vec{T}^{(\mathsf{pk})}}\right)$, where
        $\vec{N}^{(\mathsf{pk})} = (\hat{N}_i^{(\mathsf{pk})})_{i \in [5]} = (\hat{P}^x, \hat{P}^{y_1}, \hat{P}^{y_2}, \hat{P}^{z_1}, \hat{P}^{z_2})$
        $\boxed{\vec{M}^{(\mathsf{pk})} = (M_i^{(\mathsf{pk})})_{i \in [5]} = (h^{\rho_1 x}, h^{\rho_2 y_1}, h^{\rho_3 y_2}, h^{\rho_4 z_1}, h^{\rho_5 z_2})}$
        $\boxed{\vec{T}^{(\mathsf{pk})} = (T_i^{(\mathsf{pk})})_{i \in [5]} = (h^{\rho_1}, h^{\rho_2}, h^{\rho_3}, h^{\rho_4}, h^{\rho_5})}$
        $\boxed{h = H(P^{\rho_1} \| \ldots \| P^{\rho_5} \| \hat{N}_1^{(\mathsf{pk})} \| \ldots \| \hat{N}_5^{(\mathsf{pk})})}$
4 : return $(\mathsf{sk}, \mathsf{pk})$

$\mathsf{VKeyGen}(\mathsf{sk})$
1 : parse $\mathsf{sk} = (x, y_1, y_2, z_1, z_2)$
2 : $\mathsf{pk} = (\hat{X} = \hat{P}^x, \hat{Y}_1 = \hat{P}^{y_1}, \hat{Y}_2 = \hat{P}^{y_2},$
        $\hat{Z}_1 = \hat{P}^{z_1}, \hat{Z}_2 = \hat{P}^{z_2})$

$\mathsf{GenAuxTag}(\{m_1, m_2\})$
1 : $\vec{N} = (\hat{P}^{m_1}, \hat{P}^{m_2})$
2 : Sample $\rho_1, \rho_2 \xleftarrow{\$} \mathbb{Z}_p^\times$ and set $\vec{\rho} = (\rho_1, \rho_2)$
3 : $c = (P^{\rho_1} || P^{\rho_2} || N_1 || N_2)$ and $h = \mathsf{H}(c)$
4 : $\vec{M} = (h^{\rho_1 m_1}, h^{\rho_2 m_2})$, $\vec{T} = (T_1 = h^{\rho_1}, T_2 = h^{\rho_2})$
5 : return $(\vec{\rho}, \vec{T}, (\vec{M}, \vec{N}), \perp)$

$\mathsf{VerifyAux}(\mathsf{aux}, (\rho_1, \rho_2), ((M_1, M_2), (N_1, N_2)))$
1 : Compute $c = (P^{\rho_1} || P^{\rho_2} || N_1 || N_2)$
2 : Compute $(T_1, T_2) = (h^{\rho_1}, h^{\rho_2})$
3 : Compute $h := H(c)$
4 : return $\bigwedge_{i=1}^{2} e(M_i, \hat{P}) = e(h^{\rho_i}, N_i)$

$\mathsf{ConvertTag}(\vec{T} = (T_1, T_2) = (h^{\rho_1}, h^{\rho_2}))$
1 : $\gamma \xleftarrow{\$} \mathbb{Z}_p^\times$
2 : return $\vec{T}' = (T_1^\gamma, T_2^\gamma) = (h^{\rho_1 \gamma}, h^{\rho_2 \gamma})$

$\mathsf{Sign}(\mathsf{sk}, \vec{\rho}, \mathsf{aux}, (\vec{M}, \vec{N}))$
1 : parse $(\vec{M}, \vec{N}) = ((M_1, M_2), (N_1, N_2)) \in \mathcal{M}^H_{\mathsf{TDH}}$
2 : if $\mathsf{VerifyAux}(\mathsf{aux}, \vec{\rho}, (\vec{M}, \vec{N})) = 0$: return $\perp$
3 : Compute $h = \mathsf{H}(c)$
4 : $\sigma := (h, b, s) = \left(h, \prod_{j \in [2]} h^{\rho_j z_j}, h^x \prod_{j \in [2]} M_j^{y_j}\right)$
5 : return $\sigma$

$\mathsf{Verify}(\mathsf{pk}, \vec{T} = (T_1, T_2), (\vec{M}, \vec{N}), \sigma = (h, b, s))$
1 : parse $(\vec{M}, \vec{N}) = ((M_1, M_2), (N_1, N_2))$
2 : parse $\mathsf{pk} = (\vec{N}^{(\mathsf{pk})}, \boxed{\vec{M}^{(\mathsf{pk})}, \vec{T}^{(\mathsf{pk})}})$
3 : if $e(h, \hat{N}_1^{(\mathsf{pk})}) \prod_{j \in [2]} e(M_j, \hat{N}_{1+j}^{(\mathsf{pk})}) = e(s, \hat{P}) \wedge e(b, \hat{P})$
        $= \prod_{j \in [2]} e(T_j, \hat{N}_{3+j}^{(\mathsf{pk})}) \bigwedge_{j \in [2]} e(T_j, N_j) = e(M_j, \hat{P})$
4 :     return 1
5 : else return 0

$\mathsf{VerifyTag}(\vec{T}, \vec{\rho} = (\rho_1, \rho_2))$
1 : if $T_i = h^{\rho_i}$ for all $i \in \{1, 2\}$: return 1
2 : else return 0

$\mathsf{ConvertSK}(\mathsf{sk} = (x, y_1, y_2, z_1, z_2))$
1 : $\omega \xleftarrow{\$} \mathbb{Z}_p^\times$
2 : return $\mathsf{sk}' = \omega \cdot \mathsf{sk} = (\omega x, \ldots, \omega z_2)$

$\mathsf{ConvertPK}(\mathsf{pk} = (\hat{X}, \hat{Y}_1, \hat{Y}_2, \hat{Z}_1, \hat{Z}_2))$
1 : $\omega \xleftarrow{\$} \mathbb{Z}_p^\times$
2 : return $\mathsf{pk}' = \mathsf{pk}^\omega = (\hat{X}^\omega, \ldots, \hat{Z}^\omega)$

$\mathsf{ConvertSig}(\mathsf{pk}, \vec{T}, (\vec{M}, \vec{N}), \sigma)$
1 : $\omega \xleftarrow{\$} \mathbb{Z}_p^\times$
2 : parse $\sigma = (h, b, s)$
3 : return $\sigma' = (h, b^\omega, s^\omega)$ †

$\mathsf{ChangeRep}(\mathsf{pk}, \vec{T}, (\vec{M}, \vec{N}), \sigma = (h, b, s))$
1 : $(\mu, \nu) \xleftarrow{\$} (\mathbb{Z}_p^\times)^2$
2 : $\vec{T}' \leftarrow \mathsf{ConvertTag}(T, \mu)$
3 : $\sigma' = (h', b', s') = (h^{\mu\nu}, b^\mu, s^{\mu\nu})$
4 : $(\vec{M}' = \vec{M}^{\mu\upsilon}, \vec{N}' = \vec{N}^\upsilon) \in [(\vec{M}, \vec{N})]_{\mathcal{R}_{\mathsf{TDH}}}$
5 : return $(\vec{T}', (\vec{M}', \vec{N}'), \sigma')$
```

Fig. 3. Mercurial Signature (modified from [50]) scheme $\mathbf{\Pi}_{\mathsf{MBGLS}}$.

† While ConvertSig appears to only randomize the second two elements of signatures, we achieve signature unlinkability when tags are randomized in ChangeRep due to message class hiding.

Theorem 2 (Cross-Scheme Correctness). *The $\mathbf{\Pi}_{\mathsf{MBGLS}}$ construction in Figure 3 (when extended to a general ℓ by setting $n = t = 1$ in the construction in Appendix B.1) is a cross-scheme correct mercurial signature scheme scheme as per Definition 12.*

Theorem 3 (Message Class-Hiding). *Assuming the decisional Diffie-Hellman assumption is hard in the bilinear pairing groups, the $\mathbf{\Pi}_{\mathsf{MBGLS}}$ construction in Figure 3 is message class-hiding as per Definition 6.*

Theorem 4 (Tag Class-Hiding). *Assuming the decisional Diffie-Hellman assumption is hard in the bilinear pairing groups, the $\mathbf{\Pi}_{\mathsf{MBGLS}}$ construction in Figure 3 is tag class-hiding as per Definition 6.*

Theorem 5 (Origin-Hiding). *The $\mathbf{\Pi}_{\mathsf{MBGLS}}$ construction in Figure 3 is perfectly origin-hiding as per Definition 9.*

Theorem 6 (Unforgeability). *The $\mathbf{\Pi}_{\mathsf{MBGLS}}$ construction in Figure 3 is existentially unforgeable against adaptively chosen tagged message attack* ($\mathsf{EUF\text{-}CtMA}$) *as per Definition 23 (informally described in Definition 10) in the generic group model.*

Theorem 7 (Public Key Class-Hiding). *The $\mathbf{\Pi}_{\mathsf{MBGLS}}$ construction in Figure 3 is public key class-hiding as per Definition 21 (informally described in Definition 11) in the generic group model.*

For the plain version of $\mathbf{\Pi}_{\mathsf{MBGLS}}$ (without boxed elements), Theorems 1 and 3 to 7 follow from [50]. For the cross-scheme correct version of $\mathbf{\Pi}_{\mathsf{MBGLS}}$, Theorems 1 and 3 to 5 follow from [50]. Cross-scheme correctness (Theorem 2) follows from the cross-scheme correctness of our threshold mercurial signature (Theorem 9) by setting $n = t = 1$. For the cross-scheme correct version, Theorem 7 is proved in Appendix D.2, and Theorem 6 is proved in Appendix E.2.

3.3 Construction of TMS

In Figure 4, we provide a concrete construction of our threshold mercurial signature scheme $\mathsf{TMS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{GenAuxTag}, \mathsf{VerifyAux}, \mathsf{VerifyTag}, \mathsf{ParSign}, \mathsf{ParVerify}, \mathsf{Reconst}, \mathsf{Verify}, \mathsf{ConvertTag}, \mathsf{ConvertParSK}, \mathsf{ConvertParPK}, \mathsf{ConvertPK}, \mathsf{ConvertSig}, \mathsf{ChangeRep})$ (in Definition 3) for $\ell = 2$, which we call $\mathbf{\Pi}_{\mathsf{TMS}}$. We generalize this construction to $\ell \in \mathsf{poly}(\lambda)$ in Construction 1 of Appendix B.1. Note that ℓ determines the size of vectors in the scheme, so in the general case, a message-tag pair takes the form $(\vec{M}, \vec{N}) = ((M_j)_{j\in[\ell]}, (N_j)_{j\in[\ell]}), \vec{T} = (T_j)_{j\in[\ell]}$. We also present a cross-scheme correct version of $\mathbf{\Pi}_{\mathsf{TMS}}$. Elements that are required for cross-scheme correctness only (i.e. not required for Definition 3) are presented in $\boxed{\text{solid boxes}}$. We use the following parameters and building-blocks:

Parameters: The security parameter $\lambda \in \mathbb{N}$, the key size $\ell = 2$, the total number of parties $n \in \mathbb{N}$, and the signing threshold $t \in \mathbb{N}$.

Building-blocks: The Shamir secret sharing scheme $\mathsf{SSS} = (\mathsf{Share}, \mathsf{Reconst})$ as in Figure 9 and the modified mercurial signature scheme $\mathbf{\Pi}_{\mathsf{MBGLS}} = (\mathsf{Setup}$, KeyGen, $\mathsf{VKeyGen}$, $\mathsf{GenAuxTag}$, $\mathsf{VerifyAux}$, Sign, Verify, $\mathsf{VerifyTag}$, $\mathsf{ConvertTag}$, $\mathsf{ChangeRep}$, $\mathsf{ConvertSK}$, $\mathsf{ConvertPK}$, $\mathsf{ConvertSig})$ from Figure 3.

We omit the description of functions that are identical to those in Figure 3. These functions are: Setup, $\mathsf{GenAuxTag}$, $\mathsf{VerifyTag}$, Verify, $\mathsf{ConvertTag}$, $\mathsf{ConvertPK}$, $\mathsf{ConvertSig}$, and $\mathsf{ChangeRep}$. We also omit $\mathsf{ConvertParSK}$ and $\mathsf{ConvertParPK}$, which function identically as $\mathsf{ConvertSK}$ and $\mathsf{ConvertPK}$ in Figure 3, respectively.

```
$\mathsf{KeyGen}(\mathsf{pp}, n, t)$
1 : Sample $\mathsf{sk}_0 := (x, y_1, y_2, z_1, z_2) \xleftarrow{\$} \left(\mathbb{Z}_p^\times\right)^5$
2 : Sample $\boxed{\vec{\rho} = (\rho_i)_{i\in[5]}} \xleftarrow{\$} \left(\mathbb{Z}_p^\times\right)^5$
3 : Compute $\mathsf{pk}_0 = \left(\vec{N}^{(\mathsf{pk}_0)}, \boxed{\vec{M}^{(\mathsf{pk}_0)}}, \boxed{\vec{T}^{(\mathsf{pk})}}\right)$, where
      $\vec{N}^{(\mathsf{pk}_0)} = (\hat{N}_i^{(\mathsf{pk}_0)})_{i\in[5]} = (\hat{P}^{\mathsf{sk}_{0,i}})_{i\in[5]}$
      $\vec{M}^{(\mathsf{pk}_0)} = (M_i^{(\mathsf{pk}_0)})_{i\in[5]}$
                  $= (h^{\rho_1 x}, h^{\rho_2 y_1}, h^{\rho_3 y_2}, h^{\rho_4 z_1}, h^{\rho_5 z_2})$
      $\vec{T}^{(\mathsf{pk})} = (T_i^{(\mathsf{pk})})_{i\in[5]} = (h^{\rho_1}, h^{\rho_2}, h^{\rho_3}, h^{\rho_4}, h^{\rho_5})$
      $h = H(P^{\rho_1} \| \ldots \| P^{\rho_5} \| \hat{N}_1^{(\mathsf{pk}_0)} \| \ldots \| \hat{N}_5^{(\mathsf{pk}_0)})$
4 : Run $\vec{\mathsf{sk}} := (\mathsf{sk}_1, \ldots, \mathsf{sk}_n) \leftarrow \mathsf{Share}(\mathsf{sk}_0, p, n, t)$
    where $\forall i \in [n], \mathsf{sk}_i := (x_i, y_{i,1}, y_{i,2}, z_{i,1}, z_{i,2}, \boxed{\vec{\rho}})$
5 : Compute $\mathsf{pk} := (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, where
      $\mathsf{pk}_i = \left(\vec{N}^{(\mathsf{pk}_i)}, \boxed{\vec{M}^{(\mathsf{pk}_i)}}, \boxed{\vec{T}^{(\mathsf{pk})}}\right)$
      $\vec{N}^{(\mathsf{pk}_i)} = (\hat{X}_i, \hat{Y}_{i,1}, \hat{Y}_{i,2}, \hat{Z}_{i,1}, \hat{Z}_{i,2})$
                  $= (\hat{P}^{x_i}, \hat{P}^{y_{i,1}}, \hat{P}^{y_{i,2}}, \hat{P}^{z_{i,1}}, \hat{P}^{z_{i,2}})$
      $\vec{M}^{(\mathsf{pk}_i)} = (h^{\rho_1 x_i}, h^{\rho_2 y_{i,1}}, h^{\rho_3 y_{i,2}}, h^{\rho_4 z_{i,1}}, h^{\rho_5 z_{i,2}})$
    return $(\vec{\mathsf{sk}}, \vec{\mathsf{pk}}, \mathsf{pk}_0)$   // Send $(\mathsf{sk}_i, \mathsf{pk}_i)$ to $P_i$

$\mathsf{ParSign}(\mathsf{sk}_i, \vec{\rho}, \mathsf{aux}, (\vec{M}, \vec{N}))$
1 : Run $\sigma_i \leftarrow \mathbf{\Pi}_{\mathsf{MBGLS}}.\mathsf{Sign}(\mathsf{sk}_i, \vec{\rho}, \mathsf{aux}, (\vec{M}, \vec{N}))$
2 : return $\sigma_i$

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

$\mathsf{Reconst}(\vec{\mathsf{pk}}, \vec{T}, (\vec{M}, \vec{N}), \{i, \sigma_i\}_{i\in\mathfrak{T}})$
1 : if $\mathfrak{T} \not\subseteq [n] \vee |\mathfrak{T}| \neq t$ return $\perp$
2 : parse $\mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$
3 : parse $\sigma_i = (h_i, b_i, s_i)$ for $i \in \mathfrak{T}$
4 : for $i, j \in \mathfrak{T}$ do
5 :   if $h_i \neq h_j$ return $\perp$
6 : for $i \in \mathfrak{T}$ do
7 :   if $\mathsf{ParVerify}(\mathsf{pk}_i, \vec{T}, (\vec{M}, \vec{N}), \sigma_i) = 0$:
8 :     return $\perp$
9 : $\sigma := (h, b, s) = \left(h, \prod_{i\in\mathfrak{T}} b_i^{\lambda_i}, \prod_{i\in\mathfrak{T}} s_i^{\lambda_i}\right)$
10 : return $\sigma$
```

Fig. 4. Construction of threshold mercurial signature scheme $\mathbf{\Pi}_{\mathsf{TMS}}$.

TMS for arbitrary ℓ. Construction 1 in Appendix B.1 generalizes from $\ell = 2$ to an arbitrary $\ell \in \mathsf{poly}(\lambda)$. This means that, in the cross-scheme correct version, our public key takes the form $\mathsf{pk}_0 = \left(\vec{N}^{(\mathsf{pk}_0)}, \vec{M}^{(\mathsf{pk}_0)}, \vec{T}^{(\mathsf{pk})}\right)$, where:

$$\vec{N}^{(\mathsf{pk}_0)} = (\hat{N}_i^{(\mathsf{pk}_0)})_{i\in[\ell]} = (\hat{P}^x, \hat{P}^{y_1}, \ldots, \hat{P}^{y_\ell}, \hat{P}^{z_1}, \ldots, \hat{P}^{z_\ell}),$$
$$\vec{M}^{(\mathsf{pk}_0)} = (M_i^{(\mathsf{pk}_0)})_{i\in[\ell]} = (h^{\rho_1 x}, h^{\rho_2 y_1}, \ldots, h^{\rho_{2+\ell-1} y_\ell}, h^{\rho_{2+\ell} z_1}, \ldots, h^{\rho_{2+2\ell-1} z_\ell}),$$
$$\vec{T}^{(\mathsf{pk})} = (T_i^{(\mathsf{pk}_0)})_{i\in[\ell]} = (h^{\rho_1}, \ldots, h^{\rho_\ell})$$

Once we have this generalized construction, we can define how to extend the scheme so that it can be used in DAC as described in Section 3.1 in Definition 12. We show this extension ($\mathsf{ExtendSetup}$, which satisfies Definition 12) below.

– $\mathsf{ExtendSetup}(\mathsf{pp}) \to (\mathsf{pp}')$:
  1. Parse $\mathsf{pp} = (1^\lambda, \ell, (e, P, \hat{P}))$.
  2. Output $\mathsf{pp}' = (1^\lambda, \ell * 2 + 1, (e, P, \hat{P}))$.

3.4 Security Analysis of TMS

We formally state the security of $\mathbf{\Pi}_{\mathsf{TMS}}$ scheme in Theorems 8 to 14 and prove them below.

Theorem 8 (Correctness). *Assuming that the* SSS *is a correct Shamir secret sharing scheme as in Figure 9, our* $\mathbf{\Pi}_{\mathsf{TMS}}$ *construction in Figure 4 is a correct threshold mercurial signature scheme* TMS *as per Definition 4.*

Theorem 9 (Cross-Scheme Correctness). *Assuming that the* SSS *is a correct Shamir secret sharing scheme as in Figure 9, our* $\mathbf{\Pi}_{\mathsf{TMS}}$ *construction in Figure 4 (when extended to general* ℓ *in Appendix B.1) is a* cross-scheme *correct threshold mercurial signature scheme* TMS *as per Definition 12.*

Theorem 10 (Message Class-Hiding). *Assuming the decisional Diffie-Hellman assumption is hard in the bilinear pairing groups, our* $\mathbf{\Pi}_{\mathsf{TMS}}$ *construction in Figure 4 is message class-hiding as per Definition 6.*

Theorem 11 (Tag Class-Hiding). *Assuming the decisional Diffie-Hellman assumption is hard in the bilinear pairing groups, our* $\mathbf{\Pi}_{\mathsf{TMS}}$ *construction in Figure 4 is tag class-hiding as per Definition 6.*

Theorem 12 (Origin-Hiding). *Our* $\mathbf{\Pi}_{\mathsf{TMS}}$ *construction in Figure 4 is perfectly origin-hiding as per Definition 9.*

Theorem 13 (Unforgeability). *Assuming that the cross-scheme correct version of the* $\mathbf{\Pi}_{\mathsf{MBGLS}}$ *construction in Figure 3 is an unforgeable mercurial signature scheme as defined in Definition 23 (informally described in Definition 10), and assuming that* SSS *is a secure Shamir secret sharing scheme as in Figure 9, then our* $\mathbf{\Pi}_{\mathsf{TMS}}$ *construction in Figure 4 is existentially unforgeable against adaptively chosen tagged message attack* ($\mathsf{EUF\text{-}CtMA}$) *as per Definition 5 in the generic group model.*

Theorem 14 (Public Key Class-Hiding). *Assuming that the* $\mathbf{\Pi}_{\mathsf{MBGLS}}$ *construction in Figure 3 has public key class-hiding as defined in Definition 21 (informally described in Definition 11), and that* SSS *is a secure Shamir secret sharing scheme as in Figure 9, then our* $\mathbf{\Pi}_{\mathsf{TMS}}$ *construction in Figure 4 is threshold public key class-hiding as per Definition 8 in the generic group model.*

Proofs. We note that the proofs of Theorems 10 to 12 follow directly from the tag class-hiding proof of the $\mathbf{\Pi}_{\mathsf{MBGLS}}$ scheme in [50], since the construction of our messages and tags (as well as their randomization) is identical.

Proof (of Theorem 8 (Correctness)). There are five properties to the correctness definition in Definition 4. The proof of partial verification correctness is the same as the verification proof in [50] and follows from inspection. The proofs for key conversion correctness, signature conversion correctness, and change of message representative correctness also follow from inspection. Here, we provide the proof for threshold verification correctness.

For a full signature $\sigma \leftarrow \mathsf{Reconst}(\vec{\mathsf{pk}}, \vec{T}, (\vec{M}, \vec{N}), \{i, \sigma_i\}_{i\in\mathfrak{T}})$, we want to show that each condition checked by Verify passes. We know that $\sigma = (h, b, s)$, where

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

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

It follows by inspection that $e(h, \hat{X}) \prod_{j\in[2]} e(M_j, \hat{Y}_j) = e(s, \hat{P}) \wedge e(b, \hat{P}) = \prod_{j\in[2]} e(T_j, \hat{Z}_j)$. The third condition, $\bigwedge_{j\in[2]} e(T_j, N_j) = e(M_j, \hat{P})$, is true for any $(\vec{M}, \vec{N}) \in \mathcal{M}$. Therefore, $\mathsf{Verify}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma) = 1$. □

Proof (Proof of Theorem 9 (Cross-scheme Correctness)). Let $\mathsf{pp} \in \mathsf{Setup}(1^\lambda, \ell)$. By the definition of $\mathsf{ExtendSetup}$ we see that $\mathsf{pp}' \leftarrow \mathsf{ExtendSetup}(\mathsf{pp})$ is exactly the same but with $\ell' = 2\ell + 1$. Thus, any key generated by $\mathsf{KeyGen}(\mathsf{pp})$ is:

$$\begin{aligned}\mathsf{sk} =& (x, y_1, \ldots, y_\ell, z_1, \ldots, z_\ell)\\ \mathsf{pk} =& \Big((\hat{P}^x, \hat{P}^{y_1}, \ldots, \hat{P}^{y_\ell}, \hat{P}^{z_1}, \ldots, \hat{P}^{z_\ell}),\\ &(h^{\rho_x}, h^{\rho_{y,1}}, \ldots, h^{\rho_{y,\ell}}, h^{\rho_{z,1}}, \ldots, h^{\rho_{z,\ell}}),\\ &(h^{\rho_1 x}, h^{\rho_{y,1} y_1}, \ldots h^{\rho_{y,\ell} y_\ell}, h^{\rho_{z,1} z_1}, \ldots h^{\rho_{z,\ell} z_\ell})\Big)\end{aligned}$$

and any key generated by $\mathsf{KeyGen}(\mathsf{pp}')$ will be the same structure but replacing ℓ with $\ell' = 2\ell + 1$. Thus, for $\mathsf{pk} \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ and $\mathsf{pk}' \leftarrow \mathsf{KeyGen}(\mathsf{pp}')$, signature generation from the extended scheme becomes: $\sigma = \mathsf{Sign}(\mathsf{pp}', \mathsf{pk}', \vec{\rho}, \mathsf{pk}) = (h, b, s)$ where $h = H(P^{\rho_x} \| P^{\rho_{y,1}} \| \ldots \| P^{\rho_{y,\ell}} \| P^{\rho_{z,1}} \| \ldots \| P^{\rho_{z,\ell}})$, $b = \prod_{i\in[\ell]} \mathsf{pk}_{1+i}^{z'_i}$, and $s = h^{x'} \prod \mathsf{pk}_{1+\ell+i}^{y'_i}$. We can see that this passes the verification for pp': $\mathsf{Verify}(\mathsf{pp}', \mathsf{pk}', \mathsf{pk}) = e(h, \mathsf{pk}'_1) e(\mathsf{pk}_{1+\ell+i}, \mathsf{pk}'_{1+i}) = e(h^{x'} \prod \mathsf{pk}_{1+\ell+i}^{y'_i}, \hat{P}) = e(s, \hat{P})$ $\wedge\ e(\mathsf{pk}_{1+i}, \mathsf{pk}'_{1+\ell+i}) = e(\prod_{i\in[\ell]} \mathsf{pk}_{1+i}^{z'_i}, \hat{P}) = e(b, \hat{P})$. Message verification can be seen by inspection. □

We prove Theorem 13 in Appendix E.1 and Theorem 14 in Appendix D.1.

4 Threshold Delegatable Anonymous Credentials

In this section, we introduce the notion of *threshold delegatable anonymous credentials* (TDAC) with a concrete construction, which allows thresholdization. While we do not formalize revocation, our construction is compatible with the generic transformation in [46].

We outline the high-level functionality of our TDAC scheme in Section 4.1. This scheme begins with a Setup. A set of n root authorities (who can be malicious for the sake of anonymity) generate the root key pk_0 using $\mathsf{IssKeyGen}$[9] An intermediate issuer can then generate their key, pk_I, and interact with a root issuer using $\langle \mathsf{Issue} \leftrightarrow \mathsf{Receive} \rangle$ to receive a partial credential cred_i on pk_I. After interacting with with a set $\mathfrak{T}$ of root issuers (such that $|\mathfrak{T}| \geq t$), the issuer can combine the partial credential using $\mathsf{CredComb}$ and output a valid credential cred on their public key pk_I. For simplicity, intermediate issuers also use $\mathsf{IssKeyGen}$ to generate their keys just like the root issuer. A user can then be issued a credential using the $\mathsf{UIssue} \leftrightarrow \mathsf{UReceive}$ protocol and combine their credentials with the $\mathsf{UCredComb}$ protocol. A user then uses their secret key and a combined credential in an interactive protocol ($\langle \mathsf{Prove} \leftrightarrow \mathsf{Verify} \rangle$) with any verier. For simplicity, we assume that users credentials are always length L (the maximum length supported by the scheme) and issuers are always thresholdized with the same parameters, n, t, though it is easy to see how our definition and construction could be adapted to accomodate variable lengths and thresholdizing.

4.1 Syntax and Security Definition

Before we introduce our security games, we define the oracles that the adversary will use to interact with the challenger and honest users. We present these oracles in in Figure 6.

- $\underline{\mathcal{O}^{\mathsf{CreateHI}}}$: When the adversary calls $\mathcal{O}^{\mathsf{CreateHI}}$, they are indicating that the challenger should create a new honest issuer and delegate issuing power to them. The adversary specifies two sets of partial issuers (a set of issuers and a set of receivers, indicated by id_I and id_R). For example, to create the first intermediate issuer in the scheme, the adversary specifies id_I to be the handle for the root, and id_R to be a new handle that hasn't been used (which will serve as the handle for the new issuer). The oracle then proceeds with one set issuing a credential to the another set. If this handle was already used, the challenger aborts.
- $\underline{\mathcal{O}^{\mathsf{IssueToAdv}}}$: In the anonymity game, we allow the adversary to receive credentials as an issuer using the $\mathcal{O}^{\mathsf{IssueToAdv}}$ oracle.

[9] Note that, in practice, it is reasonable to assume that the key-generation (*i.e.*, $\mathsf{IssKeyGen}$) procedure is either carried out by a trusted third party or a distributed key generation (DKG) protocol.

– $\mathsf{Setup}(1^\lambda, 1^L, 1^n, 1^t) \to \mathsf{pp}$: The setup algorithm takes as input the security parameter 1^λ, level L, the total number of parties n, and the threshold t. It outputs the public parameters pp.
– $\mathsf{IssKeyGen}(\mathsf{pp}, L') \to (\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]})$: The issuer key generation algorithm takes as input the public parameter pp and level L' and outputs an issuer public key pk, and a set of partial issuer secret keys $\{\mathsf{sk}_i\}_{i\in[n]}$.
– $\mathsf{UserKeyGen}(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{sk})$: The algorithm generates a key for the user, taking as input the public parameter pp and outputting a user public key pk and a user secret key sk.
– $\langle \mathsf{Issue}(\mathsf{sk}_{\mathsf{I},i}, \mathsf{cred}_\mathsf{I}, L') \leftrightarrow \mathsf{Receive}(\mathsf{sk}_\mathsf{R}, L')\}\rangle \to \mathsf{cred}_{\mathsf{R},i}$: Credential issuance is an interactive protocol between an issuer and receiver. The credential issuance algorithm Issue takes as input the issuer partial secret key $\mathsf{sk}_{\mathsf{I},i}$, the issuer credential cred_I (which is $\perp$ for the root issuer since they are always trusted for unforgeability), and the level L'. The token receiver algorithm $\mathsf{Receive}$ takes as input receiver a partial secret key[a] sk_R, and level L'. On success, the algorithm should output a partial credential $\mathsf{cred}_{\mathsf{R},i}$ to the receiver (who can then share this with the other $(n-1)$ receivers not included in this execution of the issuance protocol).
– $\mathsf{CredComb}(\{\mathsf{cred}_{\mathsf{R},i}\}_{i\in\mathfrak{T}}, \mathsf{pk}_\mathsf{R}) \to \mathsf{cred}_\mathsf{R}$: The credential combination algorithm takes as input a set of partial credentials $\{\mathsf{cred}_{\mathsf{R},i}\}_{i\in\mathfrak{T}}$ and a user public key pk_R, and outputs an aggregated credential cred_R.
– $\langle \mathsf{UIssue}(\mathsf{sk}_{\mathsf{I},i}, \mathsf{cred}_\mathsf{I}) \leftrightarrow \mathsf{UReceive}(\mathsf{sk})\rangle \to \mathsf{cred}_\mathsf{R}$: User credential issuance is an interactive protocol between an issuer (running UIssue) and a user (running $\mathsf{UReceive}$). The issuer holds a credential cred_I at the penultimate level $L-1$ and the user holds a credential at the final level, L. The protocol results in a partial credential $\mathsf{cred}_{\mathsf{R},i}$ on the user's public key, where the user's secret key is sk.
– $\mathsf{UCredComb}(\{\mathsf{cred}_{\mathsf{R},i}\}_{i\in\mathfrak{T}}, \mathsf{pk}_\mathsf{R}) \to \mathsf{cred}_\mathsf{R}$: Combines a credential for a user (similar to $\mathsf{CredComb}$). This function takes in partial credentials $\{\mathsf{cred}_{\mathsf{R},i}\}_{i\in\mathfrak{T}}$ and combines them into a full credentials cred_R on pk_R which the user can use in the showing protocol.
– $\langle \mathsf{Prove}(\mathsf{sk}_\mathsf{P}, \mathsf{cred}_\mathsf{P}) \leftrightarrow \mathsf{Verify}(\mathsf{pk})\rangle \to 0/1$: Credential showing is an interactive protocol between a user and a credential verifier. The prover algorithm Prove takes as input a user secret sk_P and a credential cred_P, and the verification algorithm Verify takes as input the public key pk of the root authority. The result of this protocol is a bit indicating whether the showing was valid.

[a] This can be an arbitrary partial secret for the receiver.

Fig. 5. A threshold delegatable anonymous credentials scheme TDAC

– $\underline{\mathcal{O}^{\mathsf{IssueFromAdv}}}$: We allow the adversary to issue credentials to honest issuers using the $\mathcal{O}^{\mathsf{IssueFromAdv}}$ oracle[10]. In this oracle, the adversary specifies an issuer set (id_R) to receive a credential at a specified level (L').
– $\underline{\mathcal{O}^{\mathsf{IssueFromAdv}}}$: In the $\mathcal{O}^{\mathsf{IssueFromAdv}}$ oracle, the adversary receives a credential from an honest issuer set (id_I).
– $\underline{\mathcal{O}^{\mathsf{ProveToAdv}}}$: In the $\mathcal{O}^{\mathsf{ProveToAdv}}$ oracle, the adversary acts as the verifier for an honest user (id_P) who proves their credential to the adversary.
– $\underline{\mathcal{O}^{\mathsf{UIssueFromAdv}} \text{ and } \mathcal{O}^{\mathsf{UIssueToAdv}}}$: The oracles related to users, $\mathcal{O}^{\mathsf{UIssueFromAdv}}$ and $\mathcal{O}^{\mathsf{UIssueToAdv}}$, are similar to their issuer counterparts, but using the functions specified for users instead of issuers (*e.g.*, $\mathsf{UserKeyGen}$ instead of $\mathsf{IssKeyGen}$). Also, users are not thresholdized.

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

[10] Diverging from the definition in [46], we do not consider issuing forgeries (*i.e.*, when a malicious issuer in $\mathcal{O}^{\mathsf{IssueFromAdv}}$ uses a cred that they were not 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.

labeled with id. We use subscripts to distinguish these (*e.g.*, id_I for an issuer handle). To properly act as multiple honest users, the challenger keeps track of the following maps $(\mathsf{SK}, \mathsf{CRED}, \mathsf{PK})$ that map handles for honest users to their keys and credentials:

– SK holds secret keys. For issuers, SK maps handles sets of thresholdized secret keys, while for users, SK maps handles to individual secret keys.
– CRED maps handles to credentials for both issuers and users. For a handle id, the length of their credential (*i.e.*, $|\mathsf{CRED}[\mathsf{id}]|$) specifies the level in $[L]$ of the user associated with handle id.
– PK maps issuer handles to canonical representations of 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. $\mathcal{E}_{\mathcal{PK}}$ has the property that $\forall \mathsf{pk}, \in \mathcal{PK}, \mathsf{pk}' \in [\mathsf{pk}], \mathcal{E}_{\mathsf{pk}}(\mathcal{PK}) = \mathcal{E}_{\mathcal{PK}}(\mathsf{pk}')$. 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 [28]. Concretely, for our tag-based Diffie-Hellman message space, the extractor sets the first element to be the generator $N_1 = \hat{P}$ and fixes the rest of the elements accordingly so that the extracted key is in the same equivalence class: $\mathcal{E}_{\mathsf{pk}}(\vec{N}) \in [\vec{N}]_{\mathcal{R}_{\mathcal{K}}}$. We also use an extractor for credentials, $\mathcal{E}_{\mathsf{cred}}$ which extracted each public key in the showing.

Correctness. We define correctness for our threshold DAC scheme (in Definition 13).

Definition 13 (Correctness). *A threshold delegatable anonymous credentials scheme* $\mathsf{TDAC} = (\mathsf{Setup}, \mathsf{IssKeyGen}, \mathsf{UserKeyGen}, \mathsf{Issue}, \mathsf{Receive}, \mathsf{CredComb}, \mathsf{UIssue}, \mathsf{UReceive}, \mathsf{UCredComb}, \mathsf{Prove}, \mathsf{Verify})$ *is correct if for all* $\lambda \in \mathbb{N}, n \in O(\lambda), t \in O(\lambda)$ *such that* $0 < t \leq n$*, level* $L = O(\lambda)$*, issuer keys* $\{((\mathsf{sk}_{i,j}, \mathsf{pk}_{i,j})_{j\in[n]}, \mathsf{pk}_i) \in \mathsf{IssKeyGen}(\mathsf{pp}, i)\}_{i\in[L-1]}$*, partial credentials* $\{\mathsf{cred}_{i,j} = (\mathsf{Issue}(\mathsf{sk}_{i-1,j}, \mathsf{cred}_{i-1}, i-1) \leftrightarrow \mathsf{Receive}(\mathsf{sk}_i, i))\}_{j\in[n], i\in[L-1]\setminus\{0\}}$*, and credentials* $\mathsf{cred}_i = \mathsf{CredComb}(\{\mathsf{cred}_{i,j}\}_{j\in\mathfrak{T}_i})$ *for* $L-1$ *sets* $\mathfrak{T}_i \in [n]$ *such that* $|\mathfrak{T}_i| = t$*, where* $\mathsf{cred}_0 = \perp$*, user keys* $(\mathsf{sk}_L, \mathsf{pk}_L) \in \mathsf{UserKeyGen}(\mathsf{pp})$*, sets* $\mathfrak{T}, \mathfrak{T}' \in [n], |\mathfrak{T}| = |\mathfrak{T}'| = t$*, user partial credentials,* $\{\mathsf{cred}_{L,j} \in (\mathsf{UIssue}(\mathsf{sk}_{L-1,j}, \mathsf{cred}_{i-1}, i-1) \leftrightarrow \mathsf{UReceive}(\mathsf{pk}_L, i))\}_{j\in\mathfrak{T}}$*, and user combined credential* $\mathsf{cred}_L = \mathsf{UCredComb}(\{\mathsf{cred}_{L,i}\}_{i\in\mathfrak{T}})$*, it holds that* $(\mathsf{Prove}(\{\mathsf{sk}_{L,j}\}_{j\in\mathfrak{T}'}, \mathsf{cred}_L) \leftrightarrow \mathsf{Verify}(\mathsf{pk}_0)) = 1$.

Anonymity. We define anonymity for our threshold DAC scheme in Definition 14. In this game, the adversary is allowed to choose a root public key to give to the challenger, and then is allowed to interact with the challenger who plays the role of honest intermediate issuers and users. After interacting with the oracles, the adversary chooses two credentials and gives these to the challenger.

$\mathcal{O}^{\mathsf{CreateHI}}(\mathsf{id_I}, \mathfrak{T}_\mathsf{I}, \mathsf{id_R}, \mathfrak{T}_\mathsf{R}, \mathfrak{T}_\mathsf{Adv}, j \in [n], L') \to \mathsf{pk}$

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

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

```
1 : if $\mathsf{SK}[\mathsf{id_P}] = \perp \vee \mathsf{CRED}[\mathsf{id_P}] = \perp$ then return $\perp$
2 : $\mathsf{Prove}(\mathsf{SK}[\mathsf{id_P}], \mathsf{CRED}[\mathsf{id_P}]) \leftrightarrow \mathcal{A}$
3 : return $\perp$
```

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

```
1 : if $\mathsf{SK}[\mathsf{id_I}] = \perp$ then return $\perp$
2 : if $\mathsf{CRED}[\mathsf{id_I}] = \perp$ then return $\perp$
3 : if $|\mathfrak{T}_\mathsf{I}| \neq t$ then return $\perp$
4 : $(\perp, \tau) \leftarrow \left\langle \begin{matrix} \mathsf{Issue}(\mathsf{SK}[\mathsf{id_I}]_i, \mathsf{CRED}[\mathsf{id_I}], L') \\ \leftrightarrow \\ \mathcal{A} \end{matrix} \right\rangle$
5 : return $\perp$
```

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

```
1 : if $\mathsf{SK}[\mathsf{id_R}] \neq \perp$ then return $\perp$
2 : $(\mathsf{pk_R}, \mathsf{sk_R}) \leftarrow \mathsf{IssKeyGen}(\mathsf{pp})$
3 : $(\{\mathsf{cred}_i\}_{i\in\mathfrak{T}_\mathsf{R}}, \tau) \leftarrow \left\langle \begin{matrix} \mathcal{A}(\mathsf{pk_R}, i) \\ \leftrightarrow \\ \mathsf{Receive}(\mathsf{SK}[\mathsf{id_R}]_j, L') \end{matrix} \right\rangle_{i\in\mathfrak{T}_\mathsf{R}}$
4 : $\mathsf{cred} \leftarrow \mathsf{CredComb}(\{\mathsf{cred}_i\}_{i\in\mathfrak{T}_\mathsf{R}}, \mathsf{pk_R})$
5 : $\mathsf{CRED}[\mathsf{id_R}] \leftarrow \mathsf{cred}$
6 : return $\perp$
```

$\mathcal{O}^{\mathsf{UIssueFromAdv}}(\mathsf{id_R}) \leftrightarrow \mathcal{A}$

```
1 : Same as $\mathcal{O}^{\mathsf{IssueFromAdv}}$ with the following changes:
2 :   UReceive instead of Receive,
3 :   $L' = L$,     and $\mathfrak{T}_\mathsf{R} = \{1\}$
```

$\mathcal{O}^{\mathsf{UIssueToAdv}}(\mathsf{id_R}) \leftrightarrow \mathcal{A}$

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

Fig. 6. Definition of Oracles for TDAC scheme.

The challenger uses the extractor $\mathcal{E}_{\mathsf{cred}}$ on the credentials to extract a set of public keys that represent the intermediate issuers (one key for each level in $[L-1]$) and the user to which this credential was issued to. The challenger then verifies that the public keys in the this set are all honest issuers and the credentials are for honest users, aborting if this is not the case. The challenger then randomly selects one of the two users and proves possession of the corresponding credential chain to the adversary. The adversary wins if it can correctly identify which user the challenger picked (*i.e.*, the bit, b).

Definition 14 (Anonymity). *A threshold delegatable anonymous credentials scheme* TDAC = (Setup, $\mathsf{IssKeyGen}$, $\mathsf{UserKeyGen}$, Issue, $\mathsf{Receive}$, $\mathsf{CredComb}$, UIssue, $\mathsf{UReceive}$, $\mathsf{UCredComb}$, Prove, Verify) *satisfy the anonymity property if there exists a set of extractors,* $\mathcal{E} = \{\mathcal{E}_{\mathsf{pk}}, \mathcal{E}_{\mathsf{cred}}\}$*, such that for all* $\lambda \in \mathbb{N}, n \in O(\lambda), t \in O(\lambda)$ *such that* $0 < t \leq n$*, level* $L = O(\lambda)$ *and if the advantage of any*

PPT *adversaries $\mathcal{A}$ in game* $\mathsf{ANON}_{\mathcal{A}}^{\mathsf{TDAC}}(1^\lambda)$ *(defined by* $\mathsf{Adv}_{\mathsf{TDAC},\mathcal{A}}^{\mathsf{ANON}}$*) is negligible.*

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

where the anonymity game $\mathsf{ANON}_{\mathcal{A}}^{\mathsf{TDAC}}(1^\lambda)$ *is defined in Figure 7 and the oracles given to the adversary are all the oracles (* $\mathcal{O}^{\mathsf{CreateHI}}$, $\mathcal{O}^{\mathsf{CreateHU}}$, $\mathcal{O}^{\mathsf{ProveToAdv}}$, $\mathcal{O}^{\mathsf{IssueToAdv}}$, $\mathcal{O}^{\mathsf{IssueFromAdv}}$, $\mathcal{O}^{\mathsf{UIssueFromAdv}}$, *and* $\mathcal{O}^{\mathsf{UIssueToAdv}}$*) which are defined in Figure 6.*

```
$\mathsf{ANON}_{\mathcal{A}}^{\mathsf{TDAC}}(1^\lambda)$
1 : $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, 1^L, 1^n, 1^t)$
2 : $(\mathsf{pk}) \leftarrow \mathcal{A}(\mathsf{pp})$
3 : $(\mathsf{sk}_0, \mathsf{sk}_1, \mathsf{cred}_0, \mathsf{cred}_1, L') \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp})$
4 : $(\mathsf{chain}_0, \mathsf{chain}_1) \leftarrow (\mathcal{E}_{\mathsf{cred}}(\mathsf{cred}_0), \mathcal{E}_{\mathsf{cred}}(\mathsf{cred}_1))$
5 : **if** $\mathsf{chain}_0 \not\subset \mathsf{PK} \vee \mathsf{chain}_1 \not\subset \mathsf{PK}$ **then**
6 :     **return** $\perp$   // Ensure chains are honest
7 : $b \xleftarrow{\$} \{0,1\}$
8 : $\langle \mathsf{Prove}(\mathsf{sk}_b, \mathsf{cred}_b) \leftrightarrow \mathcal{A} \rangle$
9 : $b' \leftarrow \mathcal{A}_0^{\mathcal{O}}(\mathsf{pp})$
10 : **return** $b = b'$
```

```
$\mathsf{unForge}_{\mathcal{A}}^{\mathsf{TDAC}}(1^\lambda)$
1 : $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, 1^L, 1^n, 1^t)$
2 : $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i \in [n]}) \leftarrow \mathsf{IssKeyGen}(\mathsf{pp})$
3 : $b \leftarrow \langle \mathcal{A}^{\mathcal{O}_{\mathsf{Unf}}}(\mathsf{pk}) \leftrightarrow \mathsf{Verify}(\mathsf{pk}, L) \rangle$
4 : **return** $b$
5 :
```

Fig. 7. Anonymity game $\mathsf{ANON}_{\mathcal{A}}^{\mathsf{TDAC}}(1^\lambda)$ and Unforgeability game $\mathsf{unForge}_{\mathcal{A}}^{\mathsf{TDAC}}(1^\lambda)$ for TDAC scheme. The oracles $\mathcal{O}$ is described in Figure 6.

Unforgeability. We define unforgeability for our threshold DAC scheme in Definition 15. To begin the game in the unforgeability setting, an honestly generated threshold root key is generated for a known handle (*e.g.*, $\mathsf{id} = 0$), thus allowing the adversary to immediately begin calling $\mathcal{O}^{\mathsf{CreateHI}}$. The adversary is only allowed to interact with oracles that result in honest users obtaining credentials (*i.e.*, $\mathcal{O}^{\mathsf{CreateHI}}$, $\mathcal{O}^{\mathsf{CreateHU}}$, and $\mathcal{O}^{\mathsf{ProveToAdv}}$) since allowing the adversary access to the other oracles would result in the adversary being trivially able to produce a showing to defeat the game (*i.e.*, receiving a credential from $\mathcal{O}^{\mathsf{UIssueToAdv}}$).

Definition 15 (Unforgeability). *A threshold delegatable anonymous credentials scheme* $\mathsf{TDAC} = (\mathsf{Setup}, \mathsf{IssKeyGen}, \mathsf{UserKeyGen}, \mathsf{Issue}, \mathsf{Receive}, \mathsf{CredComb}, \mathsf{UIssue}, \mathsf{UReceive}, \mathsf{UCredComb}, \mathsf{Prove}, \mathsf{Verify})$ *is unforgeable if for all $\lambda \in \mathbb{N}, n \in O(\lambda), t \in O(\lambda)$ such that $0 < t \leq n$, level $L = O(\lambda)$ and if the advantage of any* PPT *adversaries $\mathcal{A}$ defined by* $\mathsf{Adv}_{\mathsf{TDAC},\mathcal{A}}^{\mathsf{unForge}}$ *in* $\mathsf{unForge}_{\mathcal{A}}^{\mathsf{TDAC}}(1^\lambda)$ *is negligible. $\mathcal{A}$ is given the oracles* $\mathcal{O}^{\mathsf{CreateHI}}$, $\mathcal{O}^{\mathsf{CreateHU}}$, *and* $\mathcal{O}^{\mathsf{ProveToAdv}}$, *from Figure 6 labeled as* $\mathcal{O}_{\mathsf{Unf}}$.

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

where the unforgeability game $\mathsf{unForge}_{\mathcal{A}}^{\mathsf{TDAC}}(1^\lambda)$ *is defined in Figure 7.*

4.2 Construction of TDAC

In Figure 8, we provide a construction of our threshold delegatable anonymous credentials scheme $\mathsf{TDAC} = (\mathsf{Setup}, \mathsf{IssKeyGen}, \mathsf{UserKeyGen}, \mathsf{Issue}, \mathsf{Receive}, \mathsf{CredComb}, \mathsf{UIssue}, \mathsf{UReceive}, \mathsf{UCredComb}, \mathsf{Prove}, \mathsf{Verify})$, which we call $\mathbf{\Pi}_{\mathsf{TDAC}}$.

Parameters: A security parameter $\lambda \in \mathbb{N}$, level $L \in \mathbb{N}$, the total number of parties that share issuer keys, $t \in \mathbb{N}$ and the threshold $t \in \mathbb{N}$.

Building-blocks: A threshold mercurial signature scheme $\mathsf{TMS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{GenAuxTag}, \mathsf{VerifyAux}, \mathsf{VerifyTag}, \mathsf{ParSign}, \mathsf{ParVerify}, \mathsf{Reconst}, \mathsf{Verify}, \mathsf{ConvertTag}, \mathsf{ConvertParSK}, \mathsf{ConvertParPK}, \mathsf{ConvertPK}, \mathsf{ConvertSig}, \mathsf{ChangeRep})$ as in Definition 3. Shamir secret sharing $\mathsf{SSS} = (\mathsf{Share}, \mathsf{Reconst})$ as in Figure 9.

4.3 Security Analysis of TDAC

We formally state the security of our threshold DAC scheme in Figure 8 in Theorems 15 to 17, which are proven in Appendix C.

Theorem 15 (Correctness). *Assume that the underlying threshold mercurial signature scheme* TMS *is correct with respect to Definition 4. Then our* $\mathbf{\Pi}_{\mathsf{TDAC}}$ *construction in Figure 8 is a correct threshold delegatable anonymous credentials scheme as per Definition 13.*

Theorem 16 (Unforgeability). *Assume that the underlying threshold mercurial signature scheme* TMS *is unforgeable with respect to Definition 5. Then our* $\mathbf{\Pi}_{\mathsf{TDAC}}$ *construction in Figure 8 is an unforgeable threshold delegatable anonymous credentials scheme* TDAC *as per Definition 15.*

Theorem 17 (Anonymity). *Assume that the underlying threshold mercurial signature scheme* TMS *is public key class hiding and message class hiding with respect to Definition 6, Definition 7, and Definition 8. Then our* $\mathbf{\Pi}_{\mathsf{TDAC}}$ *construction in Figure 8 is an anonymous threshold delegatable anonymous credentials scheme as per Definition 14.*

The threshold DAC construction $\mathbf{\Pi}_{\mathsf{TDAC}}$ is described below.

– $\mathsf{Setup}(1^\lambda, 1^L, 1^n, 1^t) \to (\mathsf{pp}, \mathsf{td})$
  1. Run $\mathsf{pp}_L \leftarrow \mathsf{TMS.Setup}(1^\lambda, \ell = 2)$
  2. For $i \in [L]$: Run $\mathsf{pp}_{L-i} \leftarrow \mathsf{TMS.ExtendSetup}(\mathsf{pp}_{L-i+1})$
  3. Output $\mathsf{pp} = (\{\mathsf{pp}_i\}_{i\in[L]}, n, t)$
– $\mathsf{IssKeyGen}(\mathsf{pp}, L') \to (\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]})$
  1. $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{TMS.KeyGen}(\mathsf{pp}_{L'}, n, t)$[a]
  2. Output $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]})$
– $\mathsf{UserKeyGen}(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{sk})$
  1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{TMS.KeyGen}(\mathsf{pp}_L, 1, 1)$
  2. Output $(\mathsf{pk}, \mathsf{sk})$
– $\langle \mathsf{Issue}(\mathsf{sk}_{\mathsf{I},i}, \mathsf{cred}_\mathsf{I}, L') \leftrightarrow \mathsf{Receive}(\mathsf{sk}_\mathsf{R}, L')\rangle \to \mathsf{cred}_{\mathsf{R},i}$:
  1. The receiver sends their combined public key, pk, to the issuer and interacts with the issuer to compute the portion of Sign that requires the tag secret $\vec{\rho}$ similar to [50][b]
  2. Issuer i then signs pk yielding a signature, $\sigma_i = \mathsf{Sign}(\mathsf{sk}_{\mathsf{I},i}, \mathsf{pk})$. The issuer sends their credential chain, $\mathsf{cred}_{L'-1}$, (which is $\perp$ for the root issuer) along with σ_i to the receiver.
  3. Receiver distributes $(\mathsf{cred}_{L'-1}, \sigma_i)$ to each other receiver.
  4. Each receiver stores this partial credential as $\mathsf{cred}_i = (\mathsf{cred}_{L'-1}, \sigma_i)$ for use in the $\mathsf{CredComb}$ function.
– $\mathsf{CredComb}(\{\mathsf{cred}_i\}_{i\in\mathfrak{T}}, \mathsf{pk}_\mathsf{R}) \to \mathsf{cred}_\mathsf{R}$:
  1. Compute the signature: $\sigma = \mathsf{TMS.CredComb}(\{\sigma_i\}_{i\in\mathfrak{T}})$ where σ_i is parsed from cred_i.
  2. Compute the credential, $\mathsf{cred} = \mathsf{cred}_{L'-1}||(\mathsf{pk}, \sigma)$ where $\mathsf{cred}_{L'-1}$ is parsed from cred_i.
– $\langle \mathsf{UIssue}(\mathsf{sk}_{\mathsf{I},i}, \mathsf{cred}_\mathsf{I}) \leftrightarrow \mathsf{UReceive}(\mathsf{sk}_\mathsf{R})\rangle \to \mathsf{cred}_\mathsf{R}$: This is exactly same as $\langle \mathsf{Issue}(\mathsf{sk}_{\mathsf{I},i}, \mathsf{cred}_\mathsf{I}, L') \leftrightarrow \{\mathsf{Receive}(\mathsf{sk}_{\mathsf{R},j}, L')\}_{j\in\mathfrak{T}}\rangle$ but with $\mathfrak{T} = \{0\}$ (only a single receiver) and $L' = L$.
– $\mathsf{UCredComb}(\{\mathsf{cred}_{\mathsf{R},i}\}_{i\in\mathfrak{T}}, \mathsf{pk}_\mathsf{R}) \to \mathsf{cred}_\mathsf{R}$: This is exactly same as $\mathsf{CredComb}(\{\mathsf{cred}_{\mathsf{R},i}\}_{i\in\mathfrak{T}}, \mathsf{pk}_\mathsf{R}) \to \mathsf{cred}_\mathsf{R}$.
– $\langle \mathsf{Prove}(\mathsf{sk}_\mathsf{P}, \mathsf{cred}_\mathsf{P}, L') \leftrightarrow \mathsf{Verify}(\mathsf{pk}, L')\rangle \to b$:
  1. The prover samples $\mu_i \xleftarrow{\$} \mathcal{MC}$, $\rho_i \xleftarrow{\$} \mathcal{KC}$ for each level ($i \in [L'] \setminus \{0\}$) in the chain and lets $\rho_0 = 1$ to ensure that the root key is not randomized.
  2. The prover randomizes all public keys and signatures in their credential cred_P using $\mathsf{pk}'_i = \mathsf{TMS.ConvertPK}(\mathsf{pk}_i; \rho_i)$, $\sigma^*_{i+1} = \mathsf{TMS.ConvertSig}(\mathsf{pk}_i, \sigma; \rho_i)$, $(\mathsf{pk}'_{i+1}, \sigma'_{i+1}) = \mathsf{TMS.ChangeRep}(\mathsf{pk}'_i, \sigma^*_i; \mu_{i+1})$ for all $i \in [L-1]$. This means that the final, $\mathsf{pk}_{L'}, \sigma_{L'}$ is randomized using $(\mathsf{pk}'_L, \sigma'_L) = \mathsf{TMS.ChangeRep}(\mathsf{pk}'_{L-1}, \sigma^*_{L-1}; \mu_L)$.
  3. The prover sends over their randomized credential, $\mathsf{cred}' = (\mathsf{cred}'_1|| \dots ||\mathsf{cred}'_L)$ where $\mathsf{cred}'_i = (\mathsf{pk}'_i, \sigma'_i)$, and performs an interactive proof of knowledge (such as a Camenisch-Stadler proof [20]) that they know the $\mu_L * \mathsf{sk}$ that corresponds to the last public key in the chain.
  4. The verifier then verifies each randomized public key in the credential with the signatures and ensures the first key is exactly pk.
  5. If all these checks hold, the verifier outputs 1 and if any checks fail, the verifier outputs 0.

[a] Since $\vec{\rho}$ is only needed to construct the initial public key (not randomizations of it) and prove the relationship between h and pk (for signing the key in $\langle \mathsf{Issue} \leftrightarrow \mathsf{Receive}\rangle$), it can be jointly computed along with this proof during DKG so that no issuer knows $\vec{\rho}$.

[b] Specifically, the receiver proves in zero-knowledge that $\vec{M}$ is correctly computed w.r.t. h such that $(\vec{M}, \vec{N}) \in \mathcal{M}$ and $\vec{T} \in \mathcal{T}^{\vec{T}}$.

Fig. 8. Construction of Threshold Delegatable Anonymous Credentials

References

1. Iso: Information technology - security techniques - anonymous digital signatures - part 2: Mechanisms using a group public key. Tech. rep., International Organization for Standardization, Geneva, Switzerland (2013), `https://www.iso.org/standard/56916.html`
2. Trusted platform module library part 1: Architecture. Tech. rep., Trusted Computing Group (2019)

3. Decentralized identifiers (dids) v1.0. Tech. rep., World Wide Web Consortium (2022)
4. Hyperledger anoncreds (2023), `https://hyperledger.github.io/anoncreds-spec/`
5. 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). https://doi.org/10.1007/978-3-642-14623-7_12
6. 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). https://doi.org/10.1007/978-3-642-22792-9_37
7. Abe, M., Haralambiev, K., Ohkubo, M.: Efficient message space extension for automorphic signatures. In: Burmester, M., Tsudik, G., Magliveras, S.S., Ilic, I. (eds.) ISC 2010. LNCS, vol. 6531, pp. 319–330. Springer, Berlin, Heidelberg (Oct 2011). https://doi.org/10.1007/978-3-642-18178-8_28
8. 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), `https://eprint.iacr.org/2024/1503`
9. 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). https://doi.org/10.1007/978-981-96-0891-1_3
10. 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). https://doi.org/10.1007/978-3-030-03329-3_14
11. 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). https://doi.org/10.1007/978-3-031-57718-5_1
12. 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), `https://github.com/eu-digital-identity-wallet/eudi-doc-architecture-and-reference-framework/discussions/211`
13. 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). https://doi.org/10.1007/978-3-642-03356-8_7
14. 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). https://doi.org/10.1007/978-3-030-92548-2_9
15. 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). https://doi.org/10.1007/978-3-662-48800-3_11

16. Camenisch, J., Herreweghen, E.V.: Design and implementation of the idemix anonymous credential system. In: Conference on Computer and Communications Security (2002), `https://api.semanticscholar.org/CorpusID:207560642`
17. 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). https://doi.org/10.1007/978-3-319-31301-6_1
18. 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). https://doi.org/10.1007/3-540-44987-6_7
19. Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 56–72. Springer, Berlin, Heidelberg (Aug 2004). https://doi.org/10.1007/978-3-540-28628-8_4
20. 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). https://doi.org/10.1007/BFb0052252
21. Celi, S., Griffy, S., Hanzlik, L., Perez-Kempner, O., Slamanig, D.: SoK: Signatures with randomizable keys. In: Clark, J., Shi, E. (eds.) FC 2024, Part II. LNCS, vol. 14745, pp. 160–187. Springer, Cham (Mar 2024). https://doi.org/10.1007/978-3-031-78679-2_9
22. 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). https://doi.org/10.1007/11818175_5
23. Chaum, D.: Security without identification: Transaction systems to make big brother obsolete. Commun. ACM **28**(10), 1030–1044 (1985). https://doi.org/10.1145/4372.4373, `https://doi.org/10.1145/4372.4373`
24. Chaum, D., van Heyst, E.: Group signatures. In: Davies, D.W. (ed.) EUROCRYPT'91. LNCS, vol. 547, pp. 257–265. Springer, Berlin, Heidelberg (Apr 1991). https://doi.org/10.1007/3-540-46416-6_22
25. Cini, V., Ramacher, S., Slamanig, D., Striecks, C., Tairi, E.: Updatable signatures and message authentication codes. In: Garay, J. (ed.) PKC 2021, Part I. LNCS, vol. 12710, pp. 691–723. Springer, Cham (May 2021). https://doi.org/10.1007/978-3-030-75245-3_25
26. Connolly, A., Lafourcade, P., Perez-Kempner, O.: Improved constructions of anonymous credentials from structure-preserving signatures on equivalence classes. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) PKC 2022, Part I. LNCS, vol. 13177, pp. 409–438. Springer, Cham (Mar 2022). https://doi.org/10.1007/978-3-030-97121-2_15
27. 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). https://doi.org/10.1007/978-981-99-8724-5_11
28. 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). https://doi.org/10.1007/978-3-030-12612-4_27
29. Crites, E.C., Lysyanskaya, A.: Mercurial signatures for variable-length messages. PoPETs **2021**(4), 441–463 (Oct 2021). https://doi.org/10.2478/popets-2021-0079

30. Das, P., Faust, S., Loss, J.: A formal treatment of deterministic wallets. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 651–668. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3354236
31. Derler, D., Slamanig, D.: Key-homomorphic signatures: definitions and applications to multiparty signatures and non-interactive zero-knowledge. DCC **87**(6), 1373–1413 (2019). https://doi.org/10.1007/s10623-018-0535-9
32. 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). https://doi.org/10.1007/3-540-48184-2_8
33. Desmedt, Y., Frankel, Y.: Threshold cryptosystems. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 307–315. Springer, New York (Aug 1990). https://doi.org/10.1007/0-387-34805-0_28
34. 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). https://doi.org/10.1109/SP46215.2023.10179470
35. Eaton, E., Stebila, D., Stracovsky, R.: Post-quantum key-blinding for authentication in anonymity networks. In: Longa, P., Ràfols, C. (eds.) LATINCRYPT 2021. LNCS, vol. 12912, pp. 67–87. Springer, Cham (Oct 2021). https://doi.org/10.1007/978-3-030-88238-9_4
36. 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. `https://ec.europa.eu/digital-building-blocks/sites/spaces/EUDIGITALIDENTITYWALLET/pages/694487738/EU+Digital+Identity+Wallet+Home` (2024), official Journal of the European Union
37. Flamini, A., Lee, E., Lysyanskaya, A.: Multi-holder anonymous credentials from BBS signatures (2025), `https://eprint.iacr.org/2024/1874`
38. Fleischhacker, N., Krupp, J., Malavolta, G., Schneider, J., Schröder, D., Simkin, M.: Efficient unlinkable sanitizable signatures from signatures with rerandomizable keys. In: Cheng, C.M., Chung, K.M., Persiano, G., Yang, B.Y. (eds.) PKC 2016, Part I. LNCS, vol. 9614, pp. 301–330. Springer, Berlin, Heidelberg (Mar 2016). https://doi.org/10.1007/978-3-662-49384-7_12
39. Fuchsbauer, G.: Automorphic signatures in bilinear groups and an application to round-optimal blind signatures. Cryptology ePrint Archive, Report 2009/320 (2009), `https://eprint.iacr.org/2009/320`
40. 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). https://doi.org/10.1007/s00145-018-9281-4
41. 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). https://doi.org/10.1007/978-3-319-39555-5_9
42. Ghadafi, E.: Short structure-preserving signatures. In: Sako, K. (ed.) CT-RSA 2016. LNCS, vol. 9610, pp. 305–321. Springer, Cham (Feb / Mar 2016). https://doi.org/10.1007/978-3-319-29485-8_18
43. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC. pp. 113–122. ACM Press (May 2008). https://doi.org/10.1145/1374376.1374396

44. Griffy, S., Lysyanskaya, A.: PACIFIC: Privacy-preserving automated contact tracing featuring integrity against cloning. CiC **1**(2), 12 (2024). https://doi.org/10.62056/ay11fhbmo
45. Griffy, S., Lysyanskaya, A., Mir, O., Kempner, O.P., Slamanig, D.: Delegatable anonymous credentials from mercurial signatures with stronger privacy. Cryptology ePrint Archive, Report 2024/1216 (2024), https://eprint.iacr.org/2024/1216
46. 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). https://doi.org/10.1007/978-981-96-0888-1_10
47. 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). https://doi.org/10.1007/978-3-540-78967-3_24
48. 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). https://doi.org/10.1007/978-3-031-14791-3_18
49. Miller, A., Xia, Y., Croman, K., Shi, E., Song, D.: The honey badger of BFT protocols. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 31–42. ACM Press (Oct 2016). https://doi.org/10.1145/2976749.2978399
50. 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). https://doi.org/10.1145/3576915.3623203
51. 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). https://doi.org/10.1109/TDSC.2023.3303834
52. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Cryptography Mailing list at https://metzdowd.com (03 2009)
53. Pan, S., Chan, K.Y., Cui, H., Yuen, T.H.: Multi-signatures for ECDSA and its applications in blockchain. In: Nguyen, K., Yang, G., Guo, F., Susilo, W. (eds.) ACISP 22. LNCS, vol. 13494, pp. 265–285. Springer, Cham (Nov 2022). https://doi.org/10.1007/978-3-031-22301-3_14
54. Paquin, C., Zaverucha, G.: U-prove cryptographic specification v1.1. Tech. rep., Microsoft Corporation (2013)
55. Putman, C., Martin, K.M.: Selective delegation of attributes in mercurial signature credentials. In: Quaglia, E.A. (ed.) 19th IMA International Conference on Cryptography and Coding. LNCS, vol. 14421, pp. 181–196. Springer, Cham (Dec 2023). https://doi.org/10.1007/978-3-031-47818-5_10
56. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979). https://doi.org/10.1145/359168.359176, https://doi.org/10.1145/359168.359176
57. 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). https://doi.org/10.1007/3-540-69053-0_18
58. 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). https://doi.org/10.14722/ndss.2019.23272

59. Yin, M., Malkhi, D., Reiter, M.K., Golan-Gueta, G., Abraham, I.: HotStuff: BFT consensus with linearity and responsiveness. In: Robinson, P., Ellen, F. (eds.) 38th ACM PODC. pp. 347–356. ACM (Jul / Aug 2019). https://doi.org/10.1145/3293611.3331591

Supplementary Material

A Additional Preliminaries

In this section, we discuss additional technical notations and cryptographic primitives relevant to our constructions and definitions.

A.1 Shamir Secret Sharing [56]

Let $P(x) = c_0 + c_1 x + \cdots + c_{t-1} x^{t-1}$ be a polynomial of degree $t-1$, where each coefficient $c_i \in \mathbb{F}$ for some finite field $\mathbb{F}$. Observe that there exists a set of scalars $\{\lambda_{\alpha,j,\mathfrak{T}}\}_{j \in [\ell]}$ such that for all $\alpha \in \mathbb{F}$ and all sets $\mathfrak{T} = \{\beta_1, \ldots, \beta_\ell\} \in \mathbb{F}^\ell$ such that $\ell \geq t$, it holds that $P(\alpha) = \sum_{j \in [\ell]} \lambda_{\alpha,j,\mathfrak{T}} P(\beta_j)$. These scalars, known as the Lagrange coefficients are computed as $\lambda_{\alpha,j,\mathfrak{T}} = \prod_{j \in \mathfrak{T}, j \neq \alpha} \frac{x - x_j}{x_\alpha - x_j}$ depends *only* on the set $\mathfrak{T}$ and the evaluation point α. Critically, the polynomial, $P(x)$, doesn't need to be known. Given the above observation, we now recall the Shamir Secret Sharing scheme [56]. Where we concretize the field to be $\mathbb{F} = \mathbb{Z}_p$ for a prime, p. Let n and t be positive integers such that $t \leq n$. A (p,n,t)-Shamir Secret Sharing ((p,n,t)-SSS or SSS for short) scheme is a pair of algorithms $(\mathsf{Share}, \mathsf{Recon})$ described below.

```
$(s_1, \ldots, s_n) \leftarrow \mathsf{Share}(s, p, n, t)$
1 : Sample $c_1, \ldots, c_t \xleftarrow{\$} \mathbb{Z}_p$
2 : Let $P(x) = s + c_1 x + \cdots + c_t x^t$
3 : for $i \in [n]$ do: Compute $s_i = P(i) \in \mathbb{Z}_p$
4 : return $(s_1, \ldots, s_n)$
```

```
$s/\perp = \mathsf{Recon}(s_{j_1}, \ldots, s_{j_\ell})$
1 : if $\ell < (t+1)$, then return $\perp$.
2 : else Compute $s = P(0) = \sum_k \lambda_{0,k,\mathfrak{T}} s_{j_k}$
3 : return $s$
```

Fig. 9. (p,n,t)-Shamir Secret Sharing scheme [56].

Security. Any (p,n,t)-SSS scheme provides the following (information-theoretic) security guarantee: for any uniformly random secret $s \xleftarrow{\$} \mathbb{Z}_p$ such that $(s_1, \ldots, s_n) \leftarrow \mathsf{Share}(s, p, n, t)$, given any subset of the shares $\mathfrak{T} \subset (s_1, \ldots, s_n)$ such that $|\mathfrak{T}| \leq t$, s remains distributed uniformly at random.

Note: In our paper, we also use the Share algorithm in the form $\mathsf{Share}(\vec{s}, p, n, t)$, where $\vec{s}$ denotes the set of elements to be shared.

A.2 Threshold signatures

Our work is closely related to [9] though we use Ghadafi SPS [42] as a starting point (specifically the mercurial variant described in [50]) instead of the signatures from [28]. Our technique improves upon [9] by not requiring interaction, though using [50] introduces more technical problems such as an interactive partial signing (to hide ρ_i) as well as it being non-trivial to extend to DAC as opposed to [28], where the bilinear pairing can simply be alternated to achieve this. We use a technique related to [27] to achieve this.

A.3 Mercurial Signatures.

The original scheme from [28] comprises the following algorithms: Setup, KeyGen, Sign, Verify, $\mathsf{ConvertPK}$, $\mathsf{ConvertSK}$, $\mathsf{ConvertSig}$, and $\mathsf{ChangeRep}$. The scheme is parametrized by a length, ℓ, which determines the upper bound on the size of messages that can be signed. A mercurial signature scheme is parameterized by equivalence relations for the message, public key, and secret key spaces: $\mathcal{R}_{\mathcal{M}}$, $\mathcal{R}_{\mathcal{K}}$, $\mathcal{R}_{\mathcal{SK}}$. These relations form *equivalence classes* for messages and keys and define exactly how messages and signatures can be randomized such that their corresponding signatures can correctly be updated to verify with the updated keys and messages. Allowing the keys and messages to be randomized is what gives this signature scheme its privacy-preserving properties.

The syntax for mercurial signatures used in [28] is given by:

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

Along with defining the functions above, a mercurial signature construction also defines the equivalence classes that are used in the correctness and security definitions. In the construction of [28], relations and equivalence classes for messages, public keys, and secret key are defined as follows:

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

$$\mathcal{R}_{\mathsf{pk}} = \{(\mathsf{pk}, \mathsf{pk}') \in (\mathbb{G}_1^*)^\ell \times (\mathbb{G}_1^*)^\ell | \exists r \in \mathbb{Z}_p^* \text{ such that } \mathsf{pk}' = \mathsf{pk}^r\}$$

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

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

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

Definition 16 (Correctness [28]). *A mercurial signature for parameterized equivalence relations, $\mathcal{R}_{\mathcal{R}}$, $\mathcal{R}_{\mathsf{pk}}$, $\mathcal{R}_{\mathsf{sk}}$, message and key randomizer spaces*[11]*, is correct if for all parameters (λ, ℓ), $\forall(\mathsf{pp}, \mathsf{td}) \in \mathsf{Setup}(1^\lambda, 1^\ell)$, and $\forall(\mathsf{sk}, \mathsf{pk}) \in \mathsf{KeyGen}(1^\lambda)$, the following holds:*

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

Definition 17 (Unforgeability [28]). *A mercurial signature scheme for parameterized equivalence relations $\mathcal{R}_{\mathcal{M}}$, $\mathcal{R}_{\mathsf{pk}}$, $\mathcal{R}_{\mathsf{sk}}$, is unforgeable if for all parameters (λ, ℓ) and all PPT adversaries* Adv*, having access to a signing oracle, there exists a negligible function* negl *such that:*

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

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

[11] We use $\mathbb{Z}_p^\times$ as our message and key randomization spaces as all known constructions use these.

Definition 18 (Message class-hiding [28]). *For all λ, ℓ and all PPT adversaries $\mathcal{A}$, there exists a negligible function* negl *such that:*

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

Definition 19 (Origin-Hiding for ConvertSig [28]). *A mercurial signature scheme is origin-hiding for* ConvertSig *if, given any tuple* (pk, σ, M) *that verifies, and given a random key randomizer* ρ, $\mathsf{ConvertSig}(\sigma, \mathsf{pk}, \rho)$ *outputs a new signature* σ' *such that* σ' *is a uniformly sampled signature in the set of verifying signatures,* $\{\sigma^* | \mathsf{Verify}(\mathsf{ConvertPK}(\mathsf{pk}, \rho), M, \sigma^*) = 1\}$.

Definition 20 (Origin-Hiding for ChangeRep [28]). *A mercurial signature scheme is origin-hiding for* ChangeRep *if, given any tuple* (pk, σ, M) *that verifies, and given a random message randomizer* μ, $\mathsf{ChangeRep}(\mathsf{pk}, M, \sigma; \mu)$ *outputs a new message and signature* M', σ' *such that* M' *is a uniform sampled message in the equivalence class of* M, $[M]_{\mathbb{R}_M}$, *and* σ' *is uniformly sampled verifying signature in the set of verifying signatures for* M', $\{\sigma^* | \mathsf{Verify}(\mathsf{pk}, M', \sigma^*) = 1\}$.

Definition 21 (Public key class-hiding [28]). *For all λ, ℓ and all PPT adversaries $\mathcal{A}$, there exists a negligible function (*negl*) such that:*

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

A.4 Aggregatable Mercurial Signatures

Here, we recall the syntax and definition of mercurial signature from [50].

Definition 22 (Aggregatable Mercurial signature syntax [50]). *A mercurial signature scheme* MS *consists of a tuple of the following* PPT *algorithms* (Setup*,* KeyGen*,* Sign*,* Verify*,* ConvertPK*,* ConvertSK*,* ConvertSig*,* ChangeRep*,* VerKey*,* VerMsg)*, which have following syntax:*

- $\mathsf{Setup}(1^\lambda, 1^\ell) \to (\mathsf{pp})$*: Outputs public parameters* pp*, including parameterized equivalence relations for the message, public key, and secret key spaces:* $\mathcal{R}_M$, $\mathcal{R}_{\mathsf{pk}}$, $\mathcal{R}_{\mathsf{sk}}$ *and the sample space for key and message converters.*
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{sk})$*: Generates an asymmetric key pair,* pk, sk.
- $\mathsf{GenAuxTag}((\vec{M}, \vec{N})) \to (\tau, \vec{T})$*: The auxiliary tag generation algorithm takes as input a message, output a tag* $\vec{T}$ *and tag secret* τ *and recompute the message to be in the equivalence class with* $\vec{T}$ *as described in Definition 2.*

- $\mathsf{VerifyTag}(\tau, \vec{T}) \to 0/1$*: The tag verification algorithm takes as input a tag* $\vec{T}$ *and its secret,* τ *and verifies that they are correlated (i.e., in the image of* $\mathsf{GenAuxTag}$ *for some message).*
- $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, M) \to \sigma$*: Signs a message* M *with the given secret key* sk.
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}, M, \sigma) \to (0 \text{ or } 1)$*: Returns 1 iff* σ *is a valid signature for* M *w.r.t.* pk.
- $\mathsf{ConvertPK}(\mathsf{pp}, \mathsf{pk}; \rho) \to \mathsf{pk}'$*: Given a key converter* ρ*, returns* pk' *by randomizing* pk *with* ρ*.*
- $\mathsf{ConvertSK}(\mathsf{pp}, \mathsf{sk}; \rho) \to \mathsf{sk}'$*: Randomize a secret key such that it now corresponds to a public key which has been randomized with the same* ρ *(i.e. signatures from* $\mathsf{sk}' = \mathsf{ConvertSK}(\mathsf{pp}, \mathsf{sk}; \rho)$ *verify by the randomized* $\mathsf{pk}' = \mathsf{ConvertPK}(\mathsf{pk}, \rho)$*).*
- $\mathsf{ConvertSig}(\mathsf{pp}, \mathsf{pk}, M, \sigma; \rho) \to \sigma'$*: Randomize the signature so that it verifies with a randomized* pk' *(which has been randomized with the same* ρ*) and* M*, but* σ' *is otherwise unlinkable to* σ*.*
- $\mathsf{ChangeRep}(\mathsf{pp}, \mathsf{pk}, M, \sigma; \mu) \to (M', \sigma')$*: Randomize the message-signature pair such that* $\mathsf{Verify}(\mathsf{pk}, M', \sigma') = 1$ *(i.e.,* σ' *and* σ *are indistinguishable) where* M' *is a new representation of the message equivalence class defined by* M*.*

The aggregatable mercurial signature unforgeability game is defined in Definition 23. We've redefined it for $j = 1$ which is what our reduction uses.

Definition 23 (Unforgeability). *An* ATMS *is unforgeable if for all PPT* $\mathcal{A}$ *having access to the oracle* $\mathcal{O}^{\mathsf{Sign}()}$ *there exists a negligible function* ϵ *s.t:* $\Pr[\mathsf{unForge}_{\mathcal{A}}^{\mathsf{ATMS}}(\lambda) = 1] \leq \epsilon(\lambda)$ *where the experiment* $\mathsf{unForge}_{\mathcal{A}}^{\mathsf{ATMS}}(\lambda)$ *is defined in Figure 10 and* Q *is the set of queries that* $\mathcal{A}$ *has issued to* $\mathcal{O}^{\mathsf{Sign}()}$*.*

```
$\underline{\mathsf{unForge}_{\mathcal{A}}^{\mathsf{ATMS}}(\lambda)}$:
  – $Q := \emptyset$; $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$;
  – $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$;
  – $(\mathsf{pk}^*, (\vec{M}^*, \vec{N}^*), \vec{T}^*, \tau^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp}, \mathsf{pk})$
  – Return:
      $\Big( \mathsf{VerifyAggr}\big(\{\mathsf{pk}\}, \vec{T}^*, \sigma^*, \{\vec{M}^*\}\big) = 1$
        $\wedge\ \mathsf{VerifyTag}(\vec{T}^*, \sigma^*, \tau^*)$
        $\wedge\ [\mathsf{pk}^*]_{\mathcal{R}_{\mathsf{pk}}} = [\mathsf{pk}]_{\mathcal{R}_{\mathsf{pk}}}$
        $\wedge\ \forall((\vec{M}, \vec{N}), \vec{T}) \in Q$ :
            $[(\vec{M}, \vec{N})]_{\mathcal{R}_{\mathsf{TDH}}} \neq [(\vec{M}^*, \vec{N}^*)]_{\mathcal{R}_{\mathsf{TDH}}}$
            $\vee\ [\vec{T}]_{\mathcal{R}_\tau} \neq [\vec{T}^*]_{\mathcal{R}_\tau} \Big)$

$\underline{\mathcal{O}^{\mathsf{Sign}}((\tau, \vec{T}), \mathsf{aux}, (\vec{M}, \vec{N}))}$:
  – $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}', \tau, \mathsf{aux}, (\vec{M}, \vec{N}))$
  – $Q = Q \cup \{(\vec{M}, \vec{N}), \vec{T}\}$,
    Return $\sigma$
```

Fig. 10. Experiment $\mathsf{unForge}_{\mathcal{A}}^{\mathsf{ATMS}}(\lambda)$

B Generalization of Π_{TMS} Construction to ℓ

In this section, we provide a threshold mercurial signature scheme construction for an arbitrary key-size ℓ with a security analysis.

We formally describe the equivalence classes in our construction below:

- For messages: $[(\vec{M}, \vec{N})]_{\mathcal{R}_{\mathcal{M}}} = \{(\vec{M}', \vec{N}') : \exists \mu, \nu \in \mathbb{Z}_p^{\times}, (\vec{M}', \vec{N}') = (\vec{M}^{\mu}, \vec{N}^{\nu})\}$
- For tags: $[\vec{T}]_{\mathcal{R}_{\mathcal{T}}} = \{\vec{T}' : \exists \gamma \in \mathbb{Z}_p^{\times}, \vec{T}' = \vec{T}^{\gamma}\}$
- For pubic keys without cross-scheme correctness: $[\vec{N}]_{\mathcal{R}_{\mathcal{K}}} = \{\vec{N}' : \exists \nu \in \mathbb{Z}_p^{\times}, \vec{N}' = \vec{N}^{\nu}\}$
- For public keys with cross-scheme correctness: $[(\vec{M}, \vec{N})]_{\mathcal{R}_{\mathcal{K}}} = \{(\vec{M}', \vec{N}') : \exists \mu, \nu \in \mathbb{Z}_p^{\times}, (\vec{M}', \vec{N}') = (\vec{M}^{\mu}, \vec{N}^{\nu})\}$
- For secret keys: $[\mathsf{sk}]_{\mathcal{R}_{\mathcal{SK}}} = \{\mathsf{sk}' : \exists \omega \in \mathbb{Z}_p^{\times}, \mathsf{sk}' = (\mathsf{sk})^{\omega}\}$
- For partial secret keys: $[\mathsf{sk}]_{\mathcal{R}_{\mathcal{PSK}}} = \{\mathsf{sk}' : \exists \omega, \nu \in \mathbb{Z}_p^{\times}, \mathsf{sk}' = (\mathsf{sk})^{\omega}\}$
- For partial pubic keys without cross-scheme correctness: $[\vec{N}]_{\mathcal{R}_{\mathcal{PK}}} = \{\vec{N}' : \exists \nu \in \mathbb{Z}_p^{\times}, \vec{N}' = \vec{N}^{\nu}\}$
- For partial public keys with cross-scheme correctness: $[(\vec{M}, \vec{N})]_{\mathcal{R}_{\mathcal{PK}}} = \{(\vec{M}', \vec{N}') : \exists \mu, \nu \in \mathbb{Z}_p^{\times}, (\vec{M}', \vec{N}') = (\vec{M}^{\mu}, \vec{N}^{\nu})\}$

B.1 Construction of TMS for arbitrary ℓ

In Figure 4 in Section 3.3, we presented a TMS construction for $\ell = 2$. Here, we revise the construction for arbitrary message and key size $\ell \in \mathsf{poly}(1^{\lambda})$ in Construction 1. Changes from Figure 4 are highlighted in blue.

Construction 1 (Threshold Mercurial Signatures for arbitrary ℓ) *We provide a concrete construction of our threshold mercurial signatures scheme* TMS = (Setup, KeyGen, GenAuxTag, VerifyAux, VerifyTag, ParSign, ParVerify, Reconst, Verify, ConvertTag, ConvertParSK, ConvertParPK, ConvertPK, ConvertSig, ChangeRep) *for arbitrary ℓ in Construction 1, which we call $\Pi^{\ell}_{\mathsf{TMS}}$. Note that the scheme can be modified to support cross-scheme correctness. Elements that are required for cross-scheme correctness only (i.e. not required for Definition 3) are presented in a $\boxed{\textit{solid box}}$.*

Parameters: *A security parameter $\lambda \in \mathbb{N}$, key size $\ell \in \mathsf{poly}(1^{\lambda})$, the total number of parties $n \in \mathbb{N}$ and the threshold $t \in \mathbb{N}$.*

Building-blocks: *Shamir secret sharing scheme* SSS = (Share, Reconst) *as in Figure 9.*

- $\mathsf{Setup}(1^{\lambda}, 1^{\ell}) \rightarrow \mathsf{pp}$:
  1. *Compute* $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, \hat{P}, e) \leftarrow \mathsf{BGGen}(1^{\lambda})$, *where p is the prime order of groups $\mathbb{G}_1 = \langle P \rangle$ and $\mathbb{G}_2 = \langle \hat{P} \rangle$, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is defined by $e(P^x, \hat{P}^y) = e(P^y, \hat{P}^x) = e(P, \hat{P})^{xy}$.*
  2. *Sample a hash function $H : \{0,1\}^* \rightarrow \mathbb{G}_1$.*

3. *Output* $\mathsf{pp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, \hat{P}, e, H)$.

- $\mathsf{KeyGen}(\mathsf{pp}, n, t) \to (\vec{\mathsf{sk}}, \vec{\mathsf{pk}}, \mathsf{pk}_0)$:
  1. *Sample* $\mathsf{sk}_0 = (x, y_1, \ldots, y_\ell, z_1, \ldots, z_\ell) \xleftarrow{\$} (\mathbb{Z}_p^\times)^\ell$ *and* $\boxed{(\rho_i)_{i\in[\ell]}} \xleftarrow{\$} (\mathbb{Z}_p^\times)^\ell$.
  2. *Compute global public key* $\mathsf{pk}_0 = \left(\vec{N}^{(\mathsf{pk}_0)}, \boxed{\vec{M}^{(\mathsf{pk}_0)}}, \boxed{\vec{T}^{(\mathsf{pk}_0)}}\right)$, *where* $\vec{N}^{(\mathsf{pk}_0)}$, $\vec{M}^{(\mathsf{pk}_0)}$, *and* $\vec{T}^{(\mathsf{pk}_0)}$ *are defined as follows:*

$$\begin{aligned}
\vec{N}^{(\mathsf{pk}_0)} &= (\hat{N}_i^{(\mathsf{pk}_0)})_{i\in[\ell]} = (\hat{P}^x, \hat{P}^{y_1}, \ldots, \hat{P}^{y_\ell}, \hat{P}^{z_1}, \ldots, \hat{P}^{z_\ell}),\\
\vec{M}^{(\mathsf{pk}_0)} &= (M_i^{(\mathsf{pk}_0)})_{i\in[\ell]} = (h^{\rho_1 x}, h^{\rho_2 y_1}, \ldots, h^{\rho_{2+\ell-1} y_\ell}, h^{\rho_{2+\ell} z_1}, \ldots, h^{\rho_{2+2\ell-1} z_\ell}),\\
\vec{T}^{(\mathsf{pk}_0)} &= (T_i^{(\mathsf{pk}_0)})_{i\in[\ell]} = (h^{\rho_1}, \ldots, h^{\rho_\ell})
\end{aligned}$$

  where $h = H(P^{\rho_1} \| \ldots \| P^{\rho_\ell} \| \hat{N}_1^{(\mathsf{pk}_0)} \| \ldots \| \hat{N}_\ell^{(\mathsf{pk}_0)})$.
  3. *Compute* $\vec{\mathsf{sk}} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_n) \leftarrow \mathsf{Share}(\mathsf{sk}_0, p, n, t)$ *where each* sk_i *for all* $i \in [n]$, *is defined as:* $\mathsf{sk}_i = (x_i, y_{i,1}, \ldots, y_{i,\ell}, z_{i,1}, \ldots, z_{i,\ell}, \vec{\rho})$.
  4. *Compute* $\vec{\mathsf{pk}} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ *such that* $\mathsf{pk}_i = \left(\vec{N}_i^{(\mathsf{pk}_i)}, \boxed{\vec{M}_i^{(\mathsf{pk}_i)}}, \boxed{\vec{T}^{(\mathsf{pk})}}\right)$, *for all* $i \in [n]$, *where* $\vec{N}_i^{(\mathsf{pk}_i)}$, *and* $\vec{M}_i^{(\mathsf{pk}_i)}$ *are defined as follows:*

$$\begin{aligned}
\vec{N}_i^{(\mathsf{pk}_i)} &= (\hat{X}_i, \hat{Y}_{i,1}, \ldots, \hat{Y}_{i,\ell}, \hat{Z}_{i,1}, \ldots, \hat{Z}_{i,\ell}) = (\hat{P}^{x_i}, \hat{P}^{y_{i,1}}, \ldots, \hat{P}^{y_{i,\ell}}, \hat{P}^{z_{i,1}}, \ldots, \hat{P}^{z_{i,\ell}})\\
\vec{M}_i^{(\mathsf{pk}_i)} &= (h^{\rho_1 x_i}, h^{\rho_2 y_{i,1}}, \ldots h^{\rho_{2+\ell-1} y_{i,\ell}}, h^{\rho_{2+\ell} z_{i,1}}, \ldots, h^{\rho_{2+2\ell-1} z_{i,\ell}})
\end{aligned}$$

  5. *Output* $(\vec{\mathsf{sk}}, \vec{\mathsf{pk}}, \mathsf{pk}_0)$ *and send each key pair* $(\mathsf{sk}_i, \mathsf{pk}_i)$ *to party* P_i.
- $\mathsf{GenAuxTag}(\vec{N}) \to (\tau, \vec{T}, c)$:
  1. *Sample* $\rho_1, \ldots, \rho_\ell \xleftarrow{\$} \mathbb{Z}_p^\times$. *Set* $c = (P^{\rho_1} \| \ldots P^{\rho_\ell} \| \vec{N})$.
  2. *Compute* $h = \mathsf{H}(c)$. *Set* $\tau = (\rho_1, \ldots, \rho_\ell)$ *and* $\vec{T} = (h^{\rho_1}, \ldots, h^{\rho_\ell})$
  3. *Output* $(\tau, \vec{T}, c)$
- $\mathsf{ParSign}(\mathsf{sk}_i, \tau, \mathsf{aux}, (\vec{M}, \vec{N})) \to \sigma_i$:
  1. *Parse* $\mathsf{sk}_i = (x_i, y_{i,1}, \ldots, y_{i,\ell}, z_{i,1}, \ldots, z_{i,\ell})$, $\tau = (\rho_1, \ldots, \rho_\ell)$, $\mathsf{aux} = (c, \bot)$, *and* $(\vec{M}, \vec{N}) = ((M_1, \ldots, M_\ell), (N_1, \ldots, N_\ell))$.
  2. *Compute* $h = H(c)$.
  3. *Compute* $\sigma_i = (h, b_i, s_i) = \left(h, \prod_{j\in[\ell]} h^{\rho_j z_{ij}}, h^{x_i} \prod_{j\in[\ell]} M_j^{y_{ij}}\right)$
  4. *Output* σ_i
- $\mathsf{ParVerify}(\mathsf{pk}_i, \vec{T}, (\vec{M}, \vec{N}), \sigma_i) \to 0/1$:
  1. *Parse* $\mathsf{pk}_i = \left(\hat{X}_i, \hat{Y}_{i,1}, \ldots, \hat{Y}_{i,\ell}, \hat{Z}_{i,1}, \ldots, \hat{Z}_{i,\ell}, \boxed{\vec{M}^{(\mathsf{pk}_i)}}, \boxed{\vec{T}^{(\mathsf{pk})}}\right)$, $\vec{T} = (T_1, \ldots, T_\ell)$, $(\vec{M}, \vec{N}) = ((M_1, \ldots, M_\ell), (N_1, \ldots, N_\ell))$, *and* $\sigma_i = (h, b_i, s_i)$.
  2. *Output* 1 *if the following holds and* 0 *otherwise:*

$$e(h, \hat{X}_i) \prod_{j\in[\ell]} e(M_j, \hat{Y}_{ij}) = e(s_i, \hat{P}) \wedge e(b_i, \hat{P}) = \prod_{j\in[\ell]} e(T_j, \hat{Z}_{ij}) \bigwedge_{j\in[\ell]} e(T_j, N_j) = e(M_j, \hat{P})$$

– $\mathsf{Reconst}(\vec{\mathsf{pk}}, \vec{T}, (\vec{M}, \vec{N}), \{i, \sigma_i\}_{i\in\mathcal{T}}) \to \sigma/\perp$:
  1. *Parse* $\vec{\mathsf{pk}} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, $(\vec{M}, \vec{N})$, *and* $\{i, \sigma_i\}_{i\in\mathcal{T}}$ *for* $\mathcal{T} \subseteq [n]$ *and* $|\mathcal{T}| = t$.
  2. *Return* $\perp$ *if* $h_i \neq h_j$ *or* $\mathsf{ParVerify}(\mathsf{pk}_i, \vec{T}, (\vec{M}, \vec{N}), \sigma_i) = 0$ *for any* $i, j \in \mathcal{T}$.
  3. *Otherwise, output the full signature as:*

$$\sigma = (h, b, s) = \left(h, \prod_{i\in\mathcal{T}} b_i^{\lambda_i}, \prod_{i\in\mathcal{T}} s_i^{\lambda_i}\right)$$

  where each λ_i *is a Lagrange coefficient computed as per the Shamir secret sharing protocol.*

– $\mathsf{Verify}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma) \to 0/1$:
  1. *Parse the global* $\mathsf{pk}_0 = \left(\hat{X}, \hat{Y}_1, \ldots, \hat{Y}_\ell, \hat{Z}_1, \ldots, \hat{Z}_\ell, \boxed{\vec{M}^{(\mathsf{pk}_0)}}, \boxed{\vec{T}^{(\mathsf{pk})}}\right)$ *and full signature* $\sigma = (h, b, s)$.
  2. *Return* 1 *if the following holds and* 0 *otherwise:*

$$e(h, \hat{X}) \prod_{j\in[\ell]} e(M_j, \hat{Y}_j) = e(s, \hat{P}) \wedge e(b, \hat{P}) = \prod_{j\in[\ell]} e(T_j, \hat{Z}_j) \bigwedge_{j\in[\ell]} e(T_j, N_j) = e(M_j, \hat{P})$$

– $\mathsf{ConvertTag}(\vec{T}, \gamma) \to \vec{T}'$:
  1. *Parse* $\vec{T} = (T_1, \ldots, T_\ell) = (h^{\rho_1}, \ldots, h^{\rho_\ell})$ *and key converter* $\gamma \in \mathbb{Z}_p^\times$.
  2. *Output* $\vec{T}' = (T_1^\gamma, \ldots, T_\ell^\gamma)$.

– $\mathsf{ConvertSK}(\mathsf{sk}_0, \omega) \to \mathsf{sk}_0'$:
  1. *Parse* $\mathsf{sk}_0 = (x, y_1, \ldots, y_\ell, z_1, \ldots, z_\ell)$ *and key converter* $\omega \in \mathbb{Z}_p^\times$.
  2. *Output* $\mathsf{sk}_0' = \omega \cdot \mathsf{sk}_0 = (\omega x, \omega y_1, \ldots, \omega y_\ell, \omega z_1, \ldots, \omega z_\ell)$.

– $\mathsf{ConvertPK}(\mathsf{pk}_0, \omega) \to \mathsf{pk}_0'$:
  1. *Parse* $\mathsf{pk}_0 = \left(\hat{X}, \hat{Y}_1, \ldots, \hat{Y}_\ell, \hat{Z}_1, \ldots, \hat{Z}_\ell, \boxed{\vec{M}^{(\mathsf{pk}_0)}}, \boxed{\vec{T}^{(\mathsf{pk})}}\right)$ *and key converter* $\omega \in \mathbb{Z}_p^\times$.
  2. *Output* $\mathsf{pk}_0' = \mathsf{pk}_0^\omega = (\hat{X}^\omega, \hat{Y}_1^\omega, \ldots, \hat{Y}_\ell^\omega, \hat{Z}_1^\omega, \ldots, \hat{Z}_\ell^\omega)$.
  3. *Output* $\mathsf{pk}_0' = \mathsf{pk}_0^\omega = \left(\hat{X}^\omega, \hat{Y}_1^\omega, \ldots, \hat{Y}_\ell^\omega, \hat{Z}_1^\omega, \ldots, \hat{Z}_\ell^\omega, \boxed{\left(\vec{M}^{(\mathsf{pk}_0)}\right)^\omega}, \boxed{\left(\vec{T}^{(\mathsf{pk})}\right)^\omega}\right)$

– $\mathsf{ConvertSig}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma, \omega) \to \sigma'$:
  1. *Parse* $\sigma = (h, b, s)$ *and key converter* $\omega \in \mathbb{Z}_p^\times$.
  2. *Output* $\sigma' = (h, b^\omega, s^\omega)$.

– $\mathsf{ChangeRep}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma, (\mu, \nu)) \to (\vec{T}', (\vec{M}', \vec{N}'), \sigma')$:
  1. *Parse* pk_0, $\vec{T} \in [\vec{T}]_{\mathcal{R}_\mathcal{T}}$, $(\vec{M}, \vec{N}) \in [(\vec{M}, \vec{N})]_{\mathcal{R}_\mathcal{M}}$, $\sigma = (h, b, s)$, *and* $\mu, \nu \in \mathbb{Z}_p^\times$.
  2. *Set* $\gamma = \mu/\nu$ *and compute* $\vec{T}' \leftarrow \mathsf{ConvertTag}(T, \gamma)$, *and set* $\vec{M}' = M^\mu$, $\vec{N}' = \vec{N}^\nu$, *and* $\sigma' = (h', b', s') = (h^\mu, b^\gamma, s^\nu)$.

3. *Output* $(\vec{T}', (\vec{M}', \vec{N}'), \sigma')$.

– $\mathsf{ExtendSetup}(\mathsf{pp}) \rightarrow (\mathsf{pp}')$*:*
  1. *Parse* $\mathsf{pp} = (1^\lambda, \ell, (e, P, \hat{P}))$.
  2. *Output* $\mathsf{pp}' = (1^\lambda, \ell * 2 + 1, (e, P, \hat{P}))$.

B.2 Security Analysis

In this section, we provide a detailed security analysis of our $\mathbf{\Pi}^\ell_{\mathsf{TMS}}$ construction. We note that the proofs of Theorems 20 to 22 follow directly from the tag class-hiding proof of the $\mathbf{\Pi}_{\mathsf{MBGLS}}$ scheme in [50], since the construction of our messages and tags (as well as their randomization) is identical.

Theorem 18 (Correctness). *Assuming that the* SSS *is a correct Shamir secret sharing scheme as in Figure 9. Then our* $\mathbf{\Pi}^\ell_{\mathsf{TMS}}$ *construction in Construction 1 is a correct threshold mercurial signature scheme* TMS *scheme as per Definition 4.*

Theorem 19 (Cross-Scheme Correctness). *Assuming that the* SSS *is a correct Shamir secret sharing scheme as in Figure 9. Then our* $\mathbf{\Pi}^\ell_{\mathsf{TMS}}$ *construction in Construction 1 is a* cross-scheme *correct hreshold mercurial signature scheme* TMS *scheme as per Definition 12.*

Theorem 20 (Message Class-Hiding). *Assuming the decisional Diffie-Hellman assumption is hard in the bilinear pairing groups. Then our* $\mathbf{\Pi}^\ell_{\mathsf{TMS}}$ *construction in Construction 1 is a message class-hiding secure threshold mercurial signature scheme* TMS *as per Definition 6.*

Theorem 21 (Tag Class-Hiding). *Assuming the decisional Diffie-Hellman assumption is hard in the bilinear pairing groups. Then our* $\mathbf{\Pi}^\ell_{\mathsf{TMS}}$ *construction in Construction 1 is a tag class-hiding secure threshold mercurial signature scheme* TMS *as per Definition 6.*

Theorem 22 (Origin-Hiding). *Our* $\mathbf{\Pi}^\ell_{\mathsf{TMS}}$ *construction in Construction 1 is an origin-hiding threshold mercurial signature threshold mercurial signature scheme* TMS *as per Definition 9.*

Theorem 23 (Unforgeability). *Assuming that the* $\mathbf{\Pi}_{\mathsf{MBGLS}}$ *construction from [50] (reviewed in Figure 3) is an unforgeable mercurial signature scheme as defined in Definition 10, and that* SSS *is a secure Shamir secret sharing scheme as in Figure 9. Then our* $\mathbf{\Pi}^\ell_{\mathsf{TMS}}$ *construction in Construction 1 is an existentially unforgeable against adaptively chosen tagged message attack* (EUF-CtMA) *secure threshold mercurial signature threshold mercurial signature scheme* TMS *as per Definition 5.*

Theorem 24 (Public Key Class-Hiding). *Assuming that the* $\mathbf{\Pi}_{\mathsf{MBGLS}}$ *construction from [50] (reviewed in Figure 3) is an unforgeable mercurial signature scheme as defined in Definition 10, and that* SSS *is a secure Shamir secret sharing scheme as in Figure 9. Then our* $\mathbf{\Pi}^\ell_{\mathsf{TMS}}$ *construction in Construction 1 is a public key class-hiding secure threshold mercurial signature threshold mercurial signature scheme* TMS *as per Definition 8.*

Proofs. The security proofs of our $\mathbf{\Pi}^{\ell}_{\mathsf{TMS}}$ construction can be proven using the same techniques employed in the security proofs of our $\mathbf{\Pi}_{\mathsf{TMS}}$ construction. Changes from proof of $\mathbf{\Pi}_{\mathsf{TMS}}$ construction are highlighted in blue.

Proof (Proof of Theorem 18). The correctness of our $\mathbf{\Pi}^{\ell}_{\mathsf{TMS}}$ construction can be proven using the same techniques employed in the proof of correctness (in Theorem 8) of our $\mathbf{\Pi}_{\mathsf{TMS}}$ construction.

By following the correctness of $\mathbf{\Pi}^{\ell}_{\mathsf{TMS}}$ construction, we provide the proof for the final property of our $\mathbf{\Pi}_{\mathsf{TMS}}$ construction, i.e., threshold verification correctness: For a full signature $\sigma \leftarrow \mathsf{Reconst}(\vec{\mathsf{pk}}, \vec{T}, (\vec{M}, \vec{N}), \{i, \sigma_i\}_{i\in\mathcal{T}})$, we want to show that each condition checked by Verify passes. We know that $\sigma = (h, b, s)$, where

$$b = \prod_{i\in\mathcal{T}} b_i^{\lambda_i} = \prod_{i\in\mathcal{T}} \prod_{j\in[\ell]} h^{\rho_j z_{i,j}\lambda_i} = \prod_{j\in[\ell]} h^{\rho_j \sum_{i\in\mathcal{T}} z_{i,j}\lambda_i} = \prod_{j\in[\ell]} h^{\rho_j z_j}$$

$$s = \prod_{i\in\mathcal{T}} s_i^{\lambda_i} = \prod_{i\in\mathcal{T}} h^{x_i\lambda_i} \prod_{j\in[\ell]} M_j^{y_{i,j}\lambda_i} = h^{\sum_{i\in\mathcal{T}} x_i\lambda_i} \prod_{j\in[\ell]} M_j^{\sum_{i\in\mathcal{T}} y_{i,j}\lambda_i} = h^x \prod_{j\in[\ell]} M_j^{y_j}$$

It follows by inspection that $e(h, \hat{X}) \prod_{j\in[\ell]} e(M_j, \hat{Y}_j) = e(s, \hat{P}) \wedge e(b, \hat{P}) = \prod_{j\in[\ell]} e(T_j, \hat{Z}_j)$. The third condition, $\bigwedge_{j\in[\ell]} e(T_j, N_j) = e(M_j, \hat{P})$, is true for any $(\vec{M}, \vec{N}) \in \mathcal{M}$. Therefore, $\mathsf{Verify}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma) = 1$.

C Security Analysis of $\mathbf{\Pi}_{\mathsf{TDAC}}$ Construction

Proof (Proof of Theorem 15). The correctness of our $\mathbf{\Pi}_{\mathsf{TDAC}}$ scheme (in Figure 8) is immediate and follows from the correctness of underlying primitive threshold mercurial signature scheme TMS.

Proof (Proof of Theorem 16). Intuitively, if an adversary defeats the unforgeability game, then they've either broken the threshold proof of knowledge scheme and reused a credential shown to them, or they've presented a credential that was never signed, and thus have broken the unforgeability (Definition 15) of the underlying mercurial signature scheme, or perhaps they've recovered the secret from the secret sharing scheme. Formally, we can create a reduction that either plays the mercurial signature unforgeability game, the knowledge extractability game of the proof system in the Prove construction, a message class hiding game, or reduces to the security of the underlying secret sharing scheme. Let these reductions be $\mathcal{R}^{\mathsf{Unf}}$, $\mathcal{R}^{\mathsf{Extract}}$, and $\mathcal{R}^{\mathsf{SS}}$. Let $\mathcal{R}^*$ be a reduction that chooses one of the 3 above reductions randomly. Let q be the number of times the adversary queries any oracle. $\mathcal{R}^{\mathsf{Unf}}$ proceeds exactly like the challenger in Definition 15 but samples $i \xleftarrow{\$} [q]$ and when the adversary makes their i-th query, if this query is to the $\mathcal{O}^{\mathsf{CreateHI}}$ oracle, the reduction uses the pk from the mercurial signature unforgeability game in Definition 5. If the adversary asks this issuer to issue to another honest issuer, the reduction uses the Sign oracle from the mercurial signature unforgeability challenger. At the end of the game, the adversary shows a credential such that for all $i \in [L-1], \mathsf{Verify}(\mathsf{pk}_i, \mathsf{pk}_{i+1}, \sigma_{i+1}) = 1$. The reduction then samples some $j \in [L-1]$ and returns $\mathsf{pk}_j, \mathsf{pk}_{j+1}, \sigma_{j+1}$ as a forgery. If pk_{j+1} is not related to an honest issuer (it is in a distinct equivalence class) then this is a valid forgery. $\mathcal{R}^{\mathsf{Extract}}$ proceeds exactly like the challenger in Definition 15 but when the adversary proves their credential at the end, the reduction outputs this to the knowledge extractability challenger. If the proof is unextractable, this reduction wins. $\mathcal{R}^{\mathsf{SS}}$ proceeds exactly like the challenger in Definition 15 but when the adversary asks for an honest issuer to be generated, the reduction uses the challenger of the hiding of the secret sharing game to generated $t-1$ shares to give to the adversary. If the adversary produces a forgery, our reduction guesses that the $t-1$ shares are simulated. Otherwise, the reduction guesses that the shares are generated legitimately from the master secret. This ensures that the adversary must either break the unforgeability of the mercurial signature scheme or the ZKP since in the third case, we're able to distinguish the secret shares. Thus, this reduction defeats once of the security games of the underlying primives with $O(1/(qL))$ chance where q and L are both polynomial.

Proof (Proof of Theorem 17). If an adversary defeats the anonymity game, it means they can distinguish two credentials. We find that these credentials are of the same length, and only contain honest users and issuers (except the root, which is the same equivalence class in both credentials since it verifies and is the scheme is unforgeable). Since each chain is honest, we can reduce to PKCH of the underlying mercurial signature scheme (Definition 8) by a number of hybrids in the length of the chain. In hybrid i, we replace the i-th issuer in the chain by a call to the PKCH challenger. In this case it will either be a distinct issuer (with a public key in a new equivalence class) or the same issuer with a differently randomized public key. After L hybrids, we find that distinguishing cred_0 from cred_1 is equivalent to distinguishing cred_0 a randomization of itself as we've replaced any distinct issuers from cred_1 with identical issuers using the hybrids. Finally, we can use origin hiding to conclude that these two games (with cred_0 and a randomization of cred_0) are indistinguishable.

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

In this section, we present the complete proofs of Theorems 14 and 7, in that order. We begin by proving the public key class-hiding (under Definition 8) of both the plan and cross-scheme correct versions of our $\mathbf{\Pi}_{\mathsf{TMS}}$ construction in Figure 4. We then prove the public key class-hiding (under Definition 11) of the cross-scheme correct version of the $\mathbf{\Pi}_{\mathsf{MBGLS}}$ construction in Figure 3. Note that the security of the plain version was already been proved in [50]).

D.1 Proof of Theorem 14

Recall that we have presented two versions of the $\mathbf{\Pi}_{\mathsf{TMS}}$ construction in Figure 4: the cross-scheme correct version that includes additional $\boxed{\text{boxed}}$ elements, and the plain version that does 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 7), 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 7. 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 14 (PK class-hiding of $\mathbf{\Pi}_{\mathsf{TMS}}$*)).* Assume for the sake of contradiction that there exists a PPT adversary $\mathcal{A}$ that wins the public key class-hiding game $\mathsf{PK\text{-}CLH}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{TMS}}}$ in Figure 2 with non-negligible advantage when executing the $\mathbf{\Pi}_{\mathsf{TMS}}$ construction from Figure 4. We construct a reduction $\mathcal{B}$ with black-box access to $\mathcal{A}$ that breaks the security of the $\mathsf{PK\text{-}CLH}_{\mathcal{B}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ game. Note that we have two cases:

1. In the proof of security of the plain $\mathbf{\Pi}_{\mathsf{TMS}}$ version, $\mathcal{B}$ plays $\mathsf{PK\text{-}CLH}_{\mathcal{B}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ against the plain version of $\mathbf{\Pi}_{\mathsf{MBGLS}}$.
2. In the proof of security of the cross-scheme correct $\mathbf{\Pi}_{\mathsf{TMS}}$ version, $\mathcal{B}$ plays $\mathsf{PK\text{-}CLH}_{\mathcal{B}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ against the cross-scheme correct version of $\mathbf{\Pi}_{\mathsf{MBGLS}}$.

Setup Phase. $\mathcal{B}$ simulates the setup phase as follows:

– $\mathcal{B}$ receives the public parameters $\mathsf{pp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, \hat{P}, e, H)$ and two sets of global public keys $\mathsf{pk}_{0,1} = (\vec{N}_1^{(\mathsf{pk}_0)}, \boxed{\vec{M}_1^{(\mathsf{pk}_0)}}, \boxed{\vec{T}_1^{(\mathsf{pk})}})$ and $\mathsf{pk}_{0,2}^{(b)} = (\vec{N}_2^{(\mathsf{pk}_0,b)}, \boxed{\vec{M}_2^{(\mathsf{pk}_0,b)}}, \boxed{\vec{T}_2^{(\mathsf{pk},b)}})$ from the challenger of $\mathsf{PK\text{-}CLH}_{\mathcal{B}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$. Note that if $b = 0$, both keys are freshly sampled, but if $b = 1$, then $\mathsf{pk}_{0,2}^{(b)}$ is a rerandomization of $\mathsf{pk}_{0,1}$.
– $\mathcal{B}$ forwards pp to $\mathcal{A}$, pretending to be the challenger of $\mathsf{PK\text{-}CLH}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{TMS}}}$, and receives in return a set of corrupted parties $\mathcal{C}$. Assume WLOG that $|\mathcal{C}| = t-1$.

(Simulating) Key Generation Phase. $\mathcal{B}$ simulates the key generation phase as follows:

– For each corrupted party P_i with index $i \in \mathcal{C}$, $\mathcal{B}$ generates the partial signing keys sk_i and partial public keys pk_i for both sets of global keys as follows:
  - $\mathcal{B}$ randomly samples signing keys $\mathsf{sk}_{1,i} = (x_i^{(1)}, y_{1,i}^{(1)}, y_{2,i}^{(1)}, z_{1,i}^{(1)}, z_{2,i}^{(1)}) \xleftarrow{\$} (\mathbb{Z}_p^{\times})^5$ and $\mathsf{sk}_{2,i}^{(b)} = (x_i^{(2,b)}, y_{1,i}^{(2,b)}, y_{2,i}^{(2,b)}, z_{1,i}^{(2,b)}, z_{2,i}^{(2,b)}) \xleftarrow{\$} (\mathbb{Z}_p^{\times})^5$.

- Compute the partial public keys $\mathsf{pk}_{1,i} = \left(\vec{N}_1^{(\mathsf{pk}_i)}, \boxed{\vec{M}_1^{(\mathsf{pk}_i)}}, \boxed{\vec{T}_1^{(\mathsf{pk})}}\right)$ and $\mathsf{pk}_{2,i}^{(b)} = \left(\vec{N}_2^{(\mathsf{pk}_i,b)}, \boxed{\vec{M}_2^{(\mathsf{pk}_i,b)}}, \boxed{\vec{T}_2^{(\mathsf{pk},b)}}\right)$, where:

$$\begin{aligned}
\vec{N}_1^{(\mathsf{pk}_i)} &= \left(\hat{X}_i^{(1)}, \hat{Y}_{1,i}^{(1)}, \hat{Y}_{2,i}^{(1)}, \hat{Z}_{1,i}^{(1)}, \hat{Z}_{2,i}^{(1)}\right) = \left(\hat{P}^{\mathsf{sk}_{i,j}^{(1)}}\right)_{j\in[5]}\\
\vec{M}_1^{(\mathsf{pk}_i)} &= \left((T_{j,1}^{(\mathsf{pk})})^{\mathsf{sk}_{i,j}^{(1)}}\right)_{j\in[5]}\\
\vec{N}_2^{(\mathsf{pk}_i,b)} &= \left(\hat{X}_i^{(2,b)}, \hat{Y}_{1,i}^{(2,b)}, \hat{Y}_{2,i}^{(2,b)}, \hat{Z}_{1,i}^{(2,b)}, \hat{Z}_{2,i}^{(2,b)}\right) = \left(\hat{P}^{\mathsf{sk}_{i,j}^{(2,b)}}\right)_{j\in[5]}\\
\vec{M}_2^{(\mathsf{pk}_i,b)} &= \left((T_{j,2}^{(\mathsf{pk},b)})^{\mathsf{sk}_{i,j}^{(2,b)}}\right)_{j\in[5]}
\end{aligned}$$

Note that $\vec{T}_1^{(\mathsf{pk})}$ and $\vec{T}_2^{(\mathsf{pk},b)}$ are the same tags outputted by the challenger in the setup phase, and thus don't need to be recomputed.

– For each honest party P_k with index $k \in \mathcal{H} = [n] \setminus \mathcal{C}$, $\mathcal{B}$ proceeds as follows:
- For all $i \in \tilde{\mathcal{T}} = \mathcal{C} \cup \{0\}$, $\mathcal{B}$ computes Lagrange coefficients evaluated at point k:

$$\tilde{\lambda}_{ki} = L_i^{\tilde{\mathcal{T}}}(k) = \prod_{j\in\tilde{\mathcal{T}} j\neq i} \frac{j-k}{j-i} \tag{1}$$

- Taking the public keys of corrupted parties $\{\mathsf{pk}_{1,i}\}_{i\in\mathcal{C}}, \{\mathsf{pk}_{2,i}^{(b)}\}_{i\in\mathcal{C}}$ and the global public keys $\mathsf{pk}_{0,1}, \mathsf{pk}_{0,2}^{(b)}$, $\mathcal{B}$ computes $\mathsf{pk}_{k,1} = (\vec{N}_1^{\mathsf{pk}_k}, \boxed{\vec{M}_1^{\mathsf{pk}_k}}, \boxed{\vec{T}_1^{\mathsf{pk}}})$ and $\mathsf{pk}_{k,2}^{(b)} = (\vec{N}_2^{\mathsf{pk}_k,b}, \boxed{\vec{M}_2^{\mathsf{pk}_k,b}}, \boxed{\vec{T}_2^{\mathsf{pk},b}})$ for all $k \in \mathcal{H}$, where:

$$\begin{aligned}
\vec{N}_1^{(\mathsf{pk}_k)} &= \left(\hat{X}_k^{(1)}, \hat{Y}_{k,1}^{(1)}, \hat{Y}_{k,2}^{(1)}, \hat{Z}_{k,1}^{(1)}, \hat{Z}_{k,2}^{(1)}\right) = \left((\hat{N}_{j,1}^{(\mathsf{pk}_0)})^{\tilde{\lambda}_{k,0}} \prod_{i\in\mathcal{C}} (\hat{N}_{j,1}^{(\mathsf{pk}_i)})^{\tilde{\lambda}_{k,i}}\right)_{j\in[5]}\\
\vec{M}_1^{(\mathsf{pk}_k)} &= \left((M_{j,1}^{(\mathsf{pk}_0)})^{\tilde{\lambda}_{k,0}} \prod_{i\in\mathcal{C}} (M_{j,1}^{(\mathsf{pk}_i)})^{\tilde{\lambda}_{k,i}}\right)_{j\in[5]}\\
\vec{N}_2^{(\mathsf{pk}_k,b)} &= \left(\hat{X}_k^{(2,b)}, \hat{Y}_{k,1}^{(2,b)}, \hat{Y}_{k,2}^{(2,b)}, \hat{Z}_{k,1}^{(2,b)}, \hat{Z}_{k,2}^{(2,b)}\right) = \left((\hat{N}_{j,2}^{(\mathsf{pk}_0,b)})^{\tilde{\lambda}_{k,0}} \prod_{i\in\mathcal{C}} (\hat{N}_{j,2}^{(\mathsf{pk}_i,b)})^{\tilde{\lambda}_{k,i}}\right)_{j\in[5]}\\
\vec{M}_2^{(\mathsf{pk}_k,b)} &= \left((M_{j,2}^{(\mathsf{pk}_0,b)})^{\tilde{\lambda}_{k,0}} \prod_{i\in\mathcal{C}} (M_{j,2}^{(\mathsf{pk}_i,b)})^{\tilde{\lambda}_{k,i}}\right)_{j\in[5]}
\end{aligned}$$

For each set of keys, $\mathcal{B}$ sends the global public key $\mathsf{pk}_{0,1}, \mathsf{pk}_{0,2}^{(b)}$, the full list of partial public keys $\vec{\mathsf{pk}}_1 = (\mathsf{pk}_{1,1}, \ldots, \mathsf{pk}_{n,1}), \vec{\mathsf{pk}}_2^{(b)} = (\mathsf{pk}_{1,2}^{(b)}, \ldots, \mathsf{pk}_{n,2}^{(b)})$, and the set of corrupted signing keys $\{\mathsf{sk}_{1,i}\}_{i\in\mathcal{C}}, \{\mathsf{sk}_{2,i}^{(b)}\}_{i\in\mathcal{C}}$ to $\mathcal{A}$.

(Simulating) Signing Phase. When $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{PSign}}(\cdot)$ with $(k, \mathsf{aux}, \vec{T}, \tau, (\vec{M}, \vec{N}))$ for an honest party index $k \in \mathcal{H}$, $\mathcal{B}$ does the following:

- $\mathcal{B}$ queries the challenger of $\mathsf{PK\text{-}CLH}_{\mathcal{B}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ for a signature on $(\vec{T}, \tau, \mathsf{aux}, (\vec{M}, \vec{N}))$ from each set of keys, and receives $\sigma_{0,1} = (h_1, b_{0,1}, s_{0,1})$ and $\sigma_{0,2}^{(b)} = (h_2^{(b)}, b_{0,2}^{(b)}, s_{0,1}^{(b)})$.
- For all corrupted parties P_i where $i \in \mathcal{C}$, since $\mathsf{sk}_{1,i}$ and $\mathsf{sk}_{2,i}^{(b)}$ are known, $\mathcal{B}$ can directly compute the partial signatures $\sigma_{1,i} = (h_1, b_{1,i}, s_{1,i}) \leftarrow \mathsf{ParSign}(\mathsf{sk}_{1,i}, \tau, \mathsf{aux}, (\vec{M}, \vec{N}))$ and $\sigma_{2,i}^{(b)} = (h_2^{(b)}, b_{2,i}^{(b)}, s_{2,i}^{(b)}) \leftarrow \mathsf{ParSign}(\mathsf{sk}_{2,i}^{(b)}, \tau, \mathsf{aux}, (\vec{M}, \vec{N}))$.
- For all $k \in \tilde{\mathcal{T}} = \mathcal{C} \cup \{0\}$, $\mathcal{B}$ computes Lagrange coefficients $\tilde{\lambda}_{ki}$ as in Equation (9).
- For all honest parties P_k where $k \in \mathcal{H}$, $\mathcal{B}$ computes the partial signatures

$$\sigma_{k,1} = (h_1, b_{k,1}, s_{k,1}) = \left(h_1, b_{0,1}^{\tilde{\lambda}_{k,0}} \prod_{i \in \mathcal{C}} b_{1,i}^{\tilde{\lambda}_{k,i}}, s_{0,1}^{\tilde{\lambda}_{k0,}} \prod_{i \in \mathcal{C}} s_{1,i}^{\tilde{\lambda}_{k,i}} \right)$$

$$\sigma_{k,2}^{(b)} = (h_2^{(b)}, b_{k,2}^{(b)}, s_{k,2}^{(b)}) = \left(h_2^{(b)}, (b_{0,2}^{(b)})^{\tilde{\lambda}_{k,0}} \prod_{i \in \mathcal{C}} (b_{2,i}^{(b)})^{\tilde{\lambda}_{k,i}}, (s_{0,2}^{(b)})^{\tilde{\lambda}_{k0,}} \prod_{i \in \mathcal{C}} (s_{2,i}^{(b)})^{\tilde{\lambda}_{k,i}} \right)$$

  and sends $\sigma_{k,1}, \sigma_{k,2}^{(b)}$ to $\mathcal{A}$.

Output Phase. At the end of the game, $\mathcal{A}$ outputs its guess b', such that $b' = b$ with non-negligible advantage. $\mathcal{B}$ forwards this guess b' to the $\mathsf{PK\text{-}CLH}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ challenger, finishing the game.

Analysis. First observe that the public parameter pp that $\mathcal{B}$ receives from the $\mathsf{PK\text{-}CLH}_{\mathcal{B}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ challenger is similarly distributed to the public parameter that $\mathcal{A}$ would receive from the *real* challenger in the $\mathsf{PK\text{-}CLH}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{TMS}}}$ game. Similarly, in the simulated key generation phase, the global public keys, both full sets of partial public keys, and both sets of corrupted signing keys that $\mathcal{B}$ generates are similarly distributed to the keys that $\mathcal{A}$ would receive from the real challenger in the $\mathsf{PK\text{-}CLH}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{TMS}}}$ game. Thus, $\mathcal{B}$ simulates $\mathsf{PK\text{-}CLH}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{TMS}}}$ exactly as in a real execution.

Now observe that, if $\mathcal{A}$'s outputs a winning guess b', then $\mathcal{B}$ will also win its game against the $\mathsf{PK\text{-}CLH}_{\mathcal{B}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ challenger. By assumption, $\mathcal{A}$ has non-negligible advantage in $\mathsf{PK\text{-}CLH}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{TMS}}}$. Thus, $\mathcal{B}$ will also win its game with non-negligible advantage. This contradicts the security of $\mathsf{PK\text{-}CLH}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ (by Theorem 7), so our initial assumption must be false.

Therefore, $\mathbf{\Pi}_{\mathsf{TMS}}$ is public key class-hiding under Definition 8. $\square$

D.2 Proof of Theorem 7

Now we provide the proof of Theorem 7, which states the public key class-hiding (under Definition 8) of the cross-scheme correct version of the $\mathbf{\Pi}_{\mathsf{MBGLS}}$ construction in Figure 3.

Proof (of Theorem 7 (PK class-hiding of $\mathbf{\Pi}_{\mathsf{MBGLS}}$*)).*
We will prove Theorem 7 in the generic group model (GGM). In addition the the signing oracle $\mathcal{O}^{\mathsf{Sign}}$ and hash oracle $\mathcal{O}^{H}$, 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 in some cases, redefine) these oracles below.

Redefining the Game. The game $\mathsf{PK\text{-}CLH}^{\mathbf{\Pi}_{\mathsf{TMS}}}_{\mathcal{A}}$ in Definition 11 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 $(\vec{T}_1, \vec{M}_1, \vec{N}_1)$ and $(\vec{T}_2^{(0)}, \vec{M}_2^{(0)}, \vec{N}_2^{(0)})$. $\mathcal{A}$ 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 $(\vec{T}_1, \vec{M}_1, \vec{N}_1)$ and $(\vec{T}_2^{(1)}, \vec{M}_2^{(1)}, \vec{N}_2^{(1)})$. $\mathcal{A}$ 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 $\mathsf{PK\text{-}CLH}^{\mathbf{\Pi}_{\mathsf{TMS}}}_{\mathcal{A}}$ 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 $(\vec{T}_1, \vec{M}_1, \vec{N}_1)$ and $(\vec{T}_2^{(0)}, \vec{M}_2^{(0)}, \vec{N}_2^{(0)})$. 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 Laurent polynomials in $\mathbb{Z}_p^{\times}[\vec{\kappa}]$. $\mathcal{A}$ 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 $(\vec{T}_1, \vec{M}_1, \vec{N}_1)$ and $(\vec{T}_2^{(1)}, \vec{M}_2^{(1)}, \vec{N}_2^{(1)})$. 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 $\mathsf{PK\text{-}CLH}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{TMS}}}$ 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 some message $(M^{(k)}, N^{(k)}, \tau^{(k)} = (\rho_1^{(k)}, \rho_2^{(k)}))$, compute the hash $h^{(k)} = H(P^{\rho_1^{(k)}} \| P^{\rho_2^{(k)}} \| \hat{N}_1^{(k)} \| \hat{N}_2^{(k)})$. If the condition $e((h^{(k)})^{\rho_j^{(k)}}, \hat{N}_j^{(k)}) = e(M_j^{(k)}, \hat{P})$ is met, output $\left(\sigma_1^{(k)} = (h^{(k)}, b_1^{(k)}, s_1^{(k)}), \sigma_2^{(b,k)} = (h^{(k)}, b_2^{(b,k)}, s_2^{(b,k)})\right)$, where

$$\begin{aligned}
h^{(k)} &= H(P^{\rho_1^{(k)}} \| P^{\rho_2^{(k)}} \| \| \hat{N}_1^{(k)} \| \hat{N}_2^{(k)}) \\
b_1^{(k)} &= \prod_{j \in [2]} (h^{(k)})^{\rho_j^{(k)} \mathsf{sk}_{1,3+j}} \\
s_1^{(k)} &= (h^{(k)})^{\mathsf{sk}_{1,1}} \prod_{j \in [2]} (M_j^{(k)})^{\mathsf{sk}_{1,1+j}} \\
b_2^{(b,k)} &= \prod_{j \in [2]} (h^{(k)})^{\rho_j^{(k)} \mathsf{sk}_{2,3+j}^{(b)}} \\
s_2^{(b,k)} &= (h^{(k)})^{\mathsf{sk}_{2,1}^{(b)}} \prod_{j \in [2]} (M_j^{(k)})^{\mathsf{sk}_{2,1+j}^{(b)}}
\end{aligned}$$

  for $b \in \{0,1\}$ (depending on which hybrid $\mathcal{A}$ is in). Else, output $\perp$. (Note that this is all in terms of indeterminants.)
- $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$, $\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}$ are GGM oracles. Each oracle takes in a set of scalars and outputs an encoding Explicitly, $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$ and $\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}$ are defined as follows:

$$\begin{aligned}
\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}} = P^{\alpha} &\prod_{i \in [5]} (M_{1,i})^{\mu_{1,i}} (T_{1,i})^{\tau_{1,i}} (M_{2,i}^{(b)})^{\mu_{2,i}} (T_{2,i}^{(b)})^{\tau_{2,i}} \prod_{i \in [q_h]} h_i^{\gamma_i} \\
&\prod_{k \in [q_\sigma]} (b_1^{(k)})^{\beta_{1,k}} (s_1^{(k)})^{\zeta_{1,k}} (b_2^{(b,k)})^{\beta_{2,k}} (s_2^{(b,k)})^{\zeta_{2,k}} \\
\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}} = \hat{P}^{\alpha} &\prod_{i \in [5]} (\hat{N}_{1,i})^{\nu_{1,i}} (\hat{N}_{2,i}^{(b)})^{\nu_{2,i}}
\end{aligned}$$

where b is either 0 or 1 depending on which hybrid we are in. Note here that $\mathsf{pk}_1 = (\vec{N}_1, \vec{M}_1, \vec{T}_1)$ and $\mathsf{pk}_2^{(b)} = (\vec{N}_2^{(b)}, \vec{M}_2^{(b)}, \vec{T}_2^{(b)})$, which uses slightly different notation than before. 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 bilinaer 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 $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}(Q_1^*)$ takes the form

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

Similarly, depending on the hybrid, the discrete log polynomial $p_2^{(*,b)}(\vec{\kappa})$ of $\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}(Q_2^*)$ takes the form

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

The combined query $\mathcal{O}_{\mathbb{G}_T}^{\mathsf{GGM}}(Q^*)$ 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 $\mathbb{Z}_p^\times[\vec{\kappa}]$.

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

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 $p(\vec{\kappa}) \in \mathbb{Z}_p^\times[\vec{\kappa}]$. In Hybrid 0, each discrete log takes a value $p(\vec{a}) \in \mathbb{Z}_p^\times$, 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 $\eta_{2,k}$ 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 $\vec{a}$. 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 7 is to prove that Hybrid 1 and Hybrid 2 are computationally indistinguishable. For this, we create a reduction to the PKCH of $\mathbf{\Pi}_{\mathsf{MBGLS}}$.

Reduction. First, we know from [45], 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 $\mathcal{A}$ to win Game 1 with non-negligible probability. We construct a reduction $\mathcal{B}$ that will run $\mathcal{A}$ as a subroutine to win Game 2 (where Game 2 is the PKCH of the construction in [50]. $\mathcal{B}$ 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 $1 + b$. As per the definition, $\mathcal{A}$ makes queries to each of $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$, $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$, $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_T}$, $\mathcal{O}^H$, and $\mathcal{O}^{\mathsf{Sign}}$. 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 $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$ 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 $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_T}$. $\mathcal{B}$ computes $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_T}(Q^*)$ 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 identically zero in one hybrid and identically nonzero in the other, the extractor has done its desired job (this has a non-negligible chance of occurring). $\mathcal{B}$ marks which hybrid $p^*(\vec{\kappa})$ is zero in, and which hybrid it is nonzero in.
   - Otherwise, $\mathcal{B}$ samples a random $b^* \overset{\$}{\leftarrow} \{0, 1\}$. $\mathcal{B}$ then starts playing Game 2 with the $\mathsf{PK\text{-}CLH}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}_{\mathcal{A}}$ challenger and immediately outputs b^*, terminating Game 2 and ending this reduction.

   In the first case, the reduction continues with the next phase.
3. **Query Phase.** $\mathcal{B}$ plays Game 2 against the $\mathsf{PK\text{-}CLH}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}_{\mathcal{A}}$ challenger, who begins by secretly choosing $b \overset{\$}{\leftarrow} \{0, 1\}$. The challenger then sends $\vec{\mathsf{pk}}_1$ and $\vec{\mathsf{pk}}_2^b$, depending on the chosen b. $\mathcal{B}$ sets $\vec{N}_1 = \vec{\mathsf{pk}}_1$ and $\vec{N}_2 = \vec{\mathsf{pk}}_2^b$. $\mathcal{B}$'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 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. (By inspection of the $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$ definition, we can see that if $\mathcal{A}$'s original query was valid, $\mathcal{B}$ is guaranteed to succeed in its computation).
   - 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 useful $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_T}$ $\mathcal{B}$ will receive will be from the distinguishing polynomial extractor $\mathcal{E}$.
   - If it is a $\mathcal{O}^{\mathsf{Sign}}$, 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 a $\mathcal{O}^{\mathsf{Sign}}$ query to the $\mathsf{PK\text{-}CLH}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}_{\mathcal{A}}$ 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 two encodings for $\mathbb{G}_2$ elements (i.e. $G_1 \| G_2 \| \hat{G}_3 \| \hat{G}_4$), then $\mathcal{B}$ tries to find the computed group element associated with each encoding. If successful, $\mathcal{B}$ sends the string of those four group elements as a $\mathcal{O}^H$ query with the $\mathsf{PK\text{-}CLH}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ 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 $\mathcal{B}$ will not be able to successfully make this $\mathcal{O}^{\mathsf{Sign}}$ query. Or, if the input to this query does *not* have the form $G_1 \| G_2 \| \hat{G}_3 \| \hat{G}_4$, $\mathcal{B}$ 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.** $\mathcal{B}$ modifies $\mathcal{E}$'s distinguishing query Q^*, setting the scalars $\mu_{1,i}$, $\tau_{1,i}$, $\mu_{2,i}$, and $\tau_{2,i}$ to 0 for all $i \in [5]$ (thereby forcibly removing any $M_{1,i}$, $T_{1,i}$, $M_{2,i}^{(b)}$, and $T_{2,i}^{(b)}$ elements from the expression). We denote this modified query by Q^{**}. Since Q^{**} is the input to a $\mathcal{O}_{\mathbb{G}_T}^{\mathsf{GGM}}$ query, it can be partitioned into the scalars Q_1^{**} corresponding to $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$ and the scalars Q_2^{**} corresponding to $\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}$. $\mathcal{B}$ attempts to compute distinguishing elements $D = \mathbb{G}_1$ 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. Then, $\mathcal{B}$ computes $e(D, \hat{D}) \in \mathbb{G}_T$:
   - If computation is unsuccessful because either $D \in \mathbb{G}_1$ or $\hat{D} \in \mathbb{G}_2$ couldn't be computed, $\mathcal{B}$ immediately outputs $\perp$ and loses Game 2. (We'll show that this won't happen).
   - If computation is successful and $e(D, \hat{D}) = e(P, \hat{P})$ (i.e. its discrete log is 0), then $\mathcal{B}$ checks if $Q^*(\kappa) = 0$. If so, $\mathcal{B}$ guesses b' ($\mathcal{A}$'s guess) as its guess. Otherwise, $\mathcal{B}$ guesses $1 - b'$.
   - If computation is successful and $e(D, \hat{D}) \neq e(P, \hat{P})$ (i.e. its discrete log is nonzero), then $\mathcal{B}$ checks if $Q^*(\kappa) \neq 0$. If so, $\mathcal{B}$ guesses b' as its guess. Otherwise, $\mathcal{B}$ guesses $1 - b'$.

   This ends the reduction to Game 2.

We make the following claims about this reduction:

Claim 3 *If $\mathcal{E}$ produces a distinguishing query Q^* whose result's discrete log polynomial is identically zero in one hybrid and identically nonzero in the other, then $\mathcal{B}$ will be able to successfully compute the modified distinguishing expression $e(D, \hat{D}) \in \mathbb{G}_T$ in the Distinguishing Phase.*

Claim 4 *If $\mathcal{E}$ produces a distinguishing query Q^* whose result's discrete log polynomial is identically zero in one hybrid and identically nonzero in the other, then with overwhelming likelihood, the discrete log polynomial $p^{**}(\vec{\kappa})$ of $e(D, \hat{D})$ will be zero when $Q^*(\kappa) = 0$ and $b' = b$ and will be non-zero when $Q^*(\kappa) \neq 0$ and $1 - b' = b$.*

If both claims are correct (and we will prove that they are), then whenever $\mathcal{E}$ produces a distinguishing query Q^* whose result's discrete log polynomial is identically zero in one hybrid and identically nonzero in the other, $\mathcal{B}$ will correctly guess the secret bit in Game 2. Otherwise, $\mathcal{B}$ has a $\frac{1}{2}$ chance of guessing correctly. Since $\mathcal{E}$ works as desired with non-negligible probability if $\mathcal{A}$ has non-negligible advantage in Game 1, then $\mathcal{B}$ will have non-negligible advantage in Game 2. This breaks the public key class-hiding of $\mathbf{\Pi}_{\mathsf{MBGLS}}$, which is a contradiction. Thus, no adversary can exist to win Game 2 with non-negligible advantage. Therefore, if we are able to prove both of these claims, we will have proven Theorem 7.

Proof of the Reduction. The proofs of these two claims will require some serious groundwork. We will begin by proving that the input $(\vec{M}^{(k)}, \vec{N}^{(k)}, \tau^{(k)})$ to some kth successful $\mathcal{O}^{\mathsf{Sign}}$ query made by $\mathcal{A}$ in Game 1 is independent of any previous $\mathcal{O}^{\mathsf{Sign}}$ queries (Lemma 1), independent of any hashes except $h^{(k)}$ (Lemma 2), independent of $\rho_{1,i}$ and $\rho_{2,i}$ for all $i \in [5]$ (Lemma 3), and able to be successfully computed by $\mathcal{B}$ in Game 2 (Lemma 4). Along the way, we will also prove that the output of a $\mathcal{O}^{\mathsf{Sign}}$ query is independent of $\rho_{1,i}$ and $\rho_{2,i}$ for all $i \in [5]$ (Corollary 1). Finally, we will prove that $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_T}(Q^*)$ is independent of any hashes except those outputted in a previous signature query (Lemma 5). All of this will allow us to prove that $e(D, \hat{D})$ can be successfully computed by $\mathcal{B}$ in Game 2 (Claim 3) and has a discrete log that is zero in some hybrid if and only if $p^*(\vec{\kappa})$ was identically zero in that hybrid (Claim 4).

Lemma 1. *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 $h^{(i)}$, $b_1^{(i)}$, $s_1^{(i)}$, $b_2^{(b,i)}$, or $s_2^{(b,i)}$ for any $i \neq k$ and $b \in \{0,1\}$, where $\left(\sigma_1^{(i)} = (h^{(i)}, b_1^{(i)}, s_1^{(i)}), \sigma_2^{(b,i)} = (h^{(i)}, b_2^{(b,i)}, s_2^{(b,i)})\right)$ 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 $h^{(i)}$, $b_1^{(i)}$, $s_1^{(i)}$, $b_2^{(b,i)}$, or $s_2^{(b,i)}$ for any $i \neq k$ and $b \in \{0,1\}$:

1. $\tau^{(k)} = (\rho_1^{(k)}, \rho_2^{(k)})$, where each $\rho_j^{(k)}$ is a scalar in $\mathbb{Z}_p^\times$. It is not possible for $\rho_j^{(k)}$ to be a function of $h^{(i)}$, $b_1^{(i)}$, $s_1^{(i)}$, $b_2^{(b,i)}$, or $s_2^{(b,i)}$, which are all elements of $\mathbb{G}_1$. 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 $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$. By definition of $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$, it is not possible for any output of this oracle to output an expression containing $h^{(i)}$, $b_1^{(i)}$, $s_1^{(i)}$, $b_2^{(b,i)}$, or $s_2^{(b,i)}$, 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 $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$. Here, we use the fact that a well-formed $\mathcal{O}^{\mathsf{Sign}}$ input must satisfy the following equation, where $h^{(k)} = H(P^{\rho_1^{(k)}} \| P^{\rho_2^{(k)}} \| \hat{N}_1^{(k)} \| \hat{N}_2^{(k)})$:

$$e((h^{(k)})^{\rho_j^{(k)}}, \hat{N}_j^{(k)}) = e(M_j^{(k)}, \hat{P}) \tag{2}$$

We can examine the discrete log of this equation:

$$\eta^{(k)}\rho_j^{(k)}p_{\hat{N}_j^{(k)}}(\vec{\kappa}) = p_{M_j^{(k)}}(\vec{\kappa}) \tag{3}$$

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, and we've just shown that $p_{\hat{N}_j^{(k)}}(\vec{\kappa})$ and $p_{M_j^{(k)}}(\vec{\kappa})$ are not functions of $h^{(i)}$, $b_1^{(i)}$, $s_1^{(i)}$, $b_2^{(b,i)}$, or $s_2^{(b,i)}$. 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 $h^{(i)}$, $b_1^{(i)}$, $s_1^{(i)}$, $b_2^{(b,i)}$, or $s_2^{(b,i)}$. 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 $h^{(i)}$, $b_1^{(i)}$, $s_1^{(i)}$, $b_2^{(b,i)}$, or $s_2^{(b,i)}$ for any $i \neq k$ and $b \in \{0, 1\}$, which concludes our proof of Lemma 3. $\square$

Lemma 2. *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 $\left(\sigma_1^{(k)} = (h^{(k)}, b_1^{(k)}, s_1^{(k)}), \sigma_2^{(b,k)} = (h^{(k)}, b_2^{(b,k)}, s_2^{(b,k)})\right)$ 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 $j \in [2]$, the condition in the following equation must be satisfied, where $h^{(k)} = H(P^{\rho_1^{(k)}} \| P^{\rho_2^{(k)}} \| \hat{N}_1^{(k)} \| \hat{N}_2^{(k)})$:

$$e((h^{(k)})^{\rho_j^{(k)}}, \hat{N}_j^{(k)}) = e(M_j^{(k)}, \hat{P}) \tag{4}$$

The discrete log of this condition is the equation

$$\eta^{(k)}\rho_j^{(k)}p_{\hat{N}_j^{(k)}}(\vec{\kappa}) = p_{M_j^{(k)}}(\vec{\kappa}) \tag{5}$$

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 $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$ and $\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}$ queries, hence our representation of their discrete logs as polynomials in $\mathbb{Z}_p^\times[\vec{\kappa}]$.

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

$$\eta^{(k)}\rho_j^{(k)}p_{\hat{N}_j^{(k)}}(\vec{\kappa}) = \gamma_i\eta_i + \cdots \tag{6}$$

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 $p_{\hat{N}_j^{(k)}}(\vec{\kappa})$ it by definition 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 2. □

Lemma 3. *If* $(M^{(k)}, N^{(k)}, \tau^{(k)})$ *is the input to a successful* $\mathcal{O}^{\mathsf{Sign}}$ *query made by* $\mathcal{A}$*, then the expressions composing* $M^{(k)}$*,* $N^{(k)}$*, and* $\tau^{(k)}$ *are not functions of* $\rho_{1,i}$ *or* $\rho_{2,i}$ *for any* $i \in [5]$*.*

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 $\rho_{1,i}$ or $\rho_{2,i}$ for any $i \in [5]$:

1. $\tau^{(k)} = (\rho_1^{(k)}, \rho_2^{(k)})$, where each $\rho_j^{(k)}$ is a scalar in $\mathbb{Z}_p^\times$. Since $\rho_{1,i}$ or $\rho_{2,i}$ are indeterminants whose value will be randomly sampled at the very end of Game 1, it is not possible for $\rho_j^{(k)}$ to be a function of these terms. Thus, $\tau^{(k)}$ is not a function of $\rho_{1,i}$ or $\rho_{2,i}$ for any $i \in [5]$.
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 the indeterminants $\rho_{1,i}$ or $\rho_{2,i}$ (since they not appear in the polynomials of any element in the second source group). Thus, $\vec{N}^{(k)}$ is not a function of $\rho_{1,i}$ or $\rho_{2,i}$ for any $i \in [5]$.
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 must satisfy the following equation, where $h^{(k)} = H(P^{\rho_1^{(k)}} \| P^{\rho_2^{(k)}} \| \hat{N}_1^{(k)} \| \hat{N}_2^{(k)})$:

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

   We can examine the discrete log of the equation above:

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

   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, and we've just shown that $p_{\hat{N}_j^{(k)}}(\vec{\kappa})$ and $p_{M_j^{(k)}}(\vec{\kappa})$ are not functions of any $\rho_{1,i}$ or $\rho_{2,i}$ for $i \in [5]$. 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 these indeterminants. Therefore, $\vec{M}^{(k)}$ is not a function of $\rho_{1,i}$ or $\rho_{2,i}$ for any $i \in [5]$.

We've shown that $M^{(k)}$, $N^{(k)}$, and $\tau^{(k)}$ all do not contain $\rho_{1,i}$ or $\rho_{2,i}$ for any $i \in [5]$, which concludes our proof of Lemma 3. □

Corollary 1. *If* $(M^{(k)}, N^{(k)}, \tau^{(k)})$ *is the input to a successful* $\mathcal{O}^{\mathsf{Sign}}$ *query made by* $\mathcal{A}$*, then the expressions composing the output* $\left(\sigma_1^{(k)} = (h^{(k)}, b_1^{(k)}, s_1^{(k)}), \sigma_2^{(b,k)} = (h^{(k)}, b_2^{(b,k)}, s_2^{(b,k)})\right)$ *are not functions of* $\rho_{1,i}$ *or* $\rho_{2,i}$ *for any* $i \in [5]$.

Proof. This follows directly from Lemma 3 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 $\left(\sigma_1^{(k)} = (h^{(k)}, b_1^{(k)}, s_1^{(k)}), \sigma_2^{(b,k)} = (h^{(k)}, b_2^{(b,k)}, s_2^{(b,k)})\right)$, where

$$\begin{aligned}
h^{(k)} &= H(P^{\rho_1^{(k)}} \| P^{\rho_2^{(k)}} \| \| \hat{N}_1^{(k)} \| \hat{N}_2^{(k)}) \\
b_1^{(k)} &= \prod_{j \in [2]} (h^{(k)})^{\rho_j^{(k)} \mathsf{sk}_{1,3+j}} \\
s_1^{(k)} &= (h^{(k)})^{\mathsf{sk}_{1,1}} \prod_{j \in [2]} (M_j^{(k)})^{\mathsf{sk}_{1,1+j}} \\
b_2^{(b,k)} &= \prod_{j \in [2]} (h^{(k)})^{\rho_j^{(k)} \mathsf{sk}_{2,3+j}^{(b)}} \\
s_2^{(b,k)} &= (h^{(k)})^{\mathsf{sk}_{2,1}^{(b)}} \prod_{j \in [2]} (M_j^{(k)})^{\mathsf{sk}_{2,1+j}^{(b)}}
\end{aligned}$$

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 $\eta^{(k)}$. The value of $\eta^{(k)}$ is randomly sampled at the very end of Game 1, so we can immediately conclude that $h^{(k)}$ is not a function of $\rho_{1,i}$ or $\rho_{2,i}$ for any $i \in [5]$.

$b_1^{(k)}$, $s_1^{(k)}$, $b_2^{(b,k)}$, and $s_2^{(b,k)}$ are composed of $h^{(k)}$, which we just showed is independent of $\rho_{1,i}$ or $\rho_{2,i}$. They are also composed of $\rho_j^{(k)}$ and $M_j^{(k)}$ for $j \in [2]$, but Lemma 3 tells us that these terms are also not functions of $\rho_{1,i}$ or $\rho_{2,i}$ for any $i \in [5]$.

Finally, the only other terms that compose $b_1^{(k)}$, $s_1^{(k)}$, $b_2^{(b,k)}$, and $s_2^{(b,k)}$ are $\mathsf{sk}_{1,j}$ and $\mathsf{sk}_{2,j}^{(b)}$ for $j \in [5]$ and $b \in \{0,1\}$. If $b = 0$, then $\mathsf{sk}_{2,j}^{(b)} = \mathsf{sk}_{2,j}$; if $b = 1$, then $\mathsf{sk}_{2,j}^{(b)} = \omega \mathsf{sk}_{1,j}$. In all cases, $\mathsf{sk}_{1,j}$, $\mathsf{sk}_{2,j}$, and ω are unique indeterminants whose values are randomly sampled at the end of Game 1, and are therefore independent of these terms are independent of $\rho_{1,i}$ and $\rho_{2,i}$ for any $i \in [5]$.

Thus, all terms in $\sigma_1^{(k)}$ and $\sigma_2^{(b,k)}$ are independent of any $\rho_{1,i}$ and $\rho_{2,i}$ for all $i \in [5]$. This concludes our proof of Corollary 1. □

Lemma 4. *If* $(M^{(k)}, N^{(k)}, \tau^{(k)})$ *is the input to a successful* $\mathcal{O}^{\mathsf{Sign}}$ *query made by* $\mathcal{A}$*, then the expressions composing* $M^{(k)}$*,* $N^{(k)}$*, and* $\tau^{(k)}$ *can all be computed by* $\mathcal{B}$ *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:

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

   $\mathcal{B}$ is given $\vec{N}_1$ and $\vec{N}_2^{(b)}$ 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:

$$P^\alpha \prod_{i\in[5]} (M_{1,i})^{\mu_{1,i}} (T_{1,i})^{\tau_{1,i}} (M_{2,i}^{(b)})^{\mu_{2,i}} (T_{2,i}^{(b)})^{\tau_{2,i}} \prod_{i\in[q_h]} h_i^{\gamma_i} \prod_{k\in[q_\sigma]} (b_1^{(k)})^{\beta_{1,k}} (s_1^{(k)})^{\zeta_{1,k}} (b_2^{(b,k)})^{\beta_{2,k}} (s_2^{(b,k)})^{\zeta_{2,k}}$$

   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 at least one of $T_{1,i}$, $M_{1,i}$, $T_{2,i}^{(b)}$, or $M_{2,i}^{(b)}$ for $i \in [5]$ and $b \in \{0,1\}$, $\mathcal{B}$ can't compute it as an element of $\mathbb{G}_1$, since these terms are not given to it by the $\mathsf{PK\text{-}CLH}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ challenger in Game 2. However, Lemma 3 tells us that the expressions composing $\vec{M}^{(k)}$ are not functions of $\rho_{1,i}$ or $\rho_{2,i}$ for any $i \in [5]$. Since each $T_{1,i}$, $M_{1,i}$, $T_{2,i}^{(b)}$, and $M_{2,i}^{(b)}$ is a function of either $\rho_{1,i}$ or $\rho_{2,i}$, then no $\vec{M}^{(k)}$ expression can contain any of these terms.
   - If the expression contains a signature component $h^{(i)}$, $b_1^{(i)}$, $s_1^{(i)}$, $b_2^{(b,i)}$, or $s_2^{(b,i)}$ 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 1 tells us that the expressions composing $M^{(k)}$ are not functions of $h^{(i)}$, $b_1^{(i)}$, $s_1^{(i)}$, $b_2^{(b,i)}$, or $s_2^{(b,i)}$ for any $i \neq k$ and $b \in \{0,1\}$, 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 2 tells us that the expressions composing $M^{(k)}$ are not functions of any hash h_i except $h_i = h^{(k)} = H(P^{\rho_1^{(k)}} \| P^{\rho_2^{(k)}} \| \hat{N}_1^{(k)} \| \hat{N}_2^{(k)})$. We've just shown that each $\rho_j^{(k)}$ and $\hat{N}_j^{(k)}$ for $j \in [2]$ can be computed by $\mathcal{B}$, so this hash $h^{(k)}$ will be computable.

   Thus, it is guaranteed that $\mathcal{B}$ will always be able to compute $\vec{M}^*$ 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 4. □

Lemma 5. *If $\mathcal{E}$ produces a distinguishing query Q^* such that the discrete log polynomial $p^*(\vec{\kappa})$ of $\mathcal{O}_{\mathbb{G}_T}^{\mathsf{GGM}}(Q^*)$ is identically zero in one hybrid and identically nonzero in the other, 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 $\left(\sigma_1^{(k)} = (h^{(k)}, b_1^{(k)}, s_1^{(k)}), \sigma_2^{(b,k)} = (h^{(k)}, b_2^{(b,k)}, s_2^{(b,k)})\right)$ of some $\mathcal{O}^{\mathsf{Sign}}$ query.

Proof. For ease of presentation, we can partition the scalars of Q^* into scalars for $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$ and scalars for $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$. Denote these sets by Q_1^* and Q_2^*.

Let $b = 0$ denote that $\mathcal{A}$ is in Hybrid 1, and $b = 1$ denote that $\mathcal{A}$ is in Hybrid 2. Depending on the hybrid, the discrete log polynomial $p_1^{(*,b)}(\vec{\kappa})$ of $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}(Q_1^*)$ takes the form

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

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

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

The combined query $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_T}(Q^*)$ has discrete log $p^{(*,b)}(\vec{\kappa}) = p_1^{(*,b)}(\vec{\kappa}) \cdot p_2^{(*,b)}(\vec{\kappa})$.

Assume toward contradiction that for some h_i that is the result of a $\mathcal{O}^H$ query but not of the form $h_i = h^{(k)}$, the associated scalar γ_i is nonzero in Q^*. By compouting this product, we can see that the polynomial $p^{*,b}(\vec{\kappa})$ will contain the terms

$$\gamma_i(\eta_i)\left(\alpha_2 + \sum_{j\in[5]}(\nu_{1,j}(\mathsf{sk}_{1,j}) + \nu_{2,j}(\mathsf{sk}_{2,j}))\right)$$
$$\gamma_i(\eta_i)\left(\alpha_2 + \sum_{j\in[5]}(\nu_{1,j}(\mathsf{sk}_{1,j}) + \nu_{2,j}(\omega\mathsf{sk}_{1,j}))\right)$$

depending on whether $b = 0$ or $b = 1$, respectively. If $\gamma_i \neq$ and no other term contains $\eta_i\mathsf{sk}_{*,j}$ to "cancel out" this term, we see that these terms will be nonzero in both hybrids. This would be a contradiction, as we've assumed that $\mathcal{E}$ extracted a Q^* such that one of $p^{(*,0)}(\vec{\kappa})$ or $p^{(*,1)}(\vec{\kappa})$ is 0. If we can show that no other term in $p^{(*,b)}(\vec{\kappa})$ contains the indeterminant $\eta_i\mathsf{sk}_{*,j}$, we can show that this sum cannot be canceled out, no matter what other scalars are chosen in Q^*.

1. Any other h_j outputted by $\mathcal{O}^H$ for $i \neq j \in [q_h]$ has a discrete log $\eta_j \neq \eta_i$. Thus, no other γ_j scalar can be used to cancel out this sum.
2. Any indeterminant $\eta^{(k)}$ for $k \in [q_\sigma]$ is by our assumption distinct from η_i.
3. Lemma 2 tells us that $\rho_j^{(k)}$ and $p_{M_j^{(k)}}(\vec{\kappa})$ are independent of any hash not of the form $\eta^{(k)}$, so these cannot include η_i.

The remaining terms in $p^{(*,b)}(\vec{\kappa})$ can be ruled out by simple inspection. Thus, we've shown that there is no way to cancel out the sum. As long as γ_i is nonzero, both $p^{(*,0)}(\vec{\kappa})$ and $p^{(*,1)}(\vec{\kappa})$ will be nonzero. This is a contradiction, so it must be the case that if h_i is not of the form $h_i = h^{(k)}$ for $k \in [q_h]$, the associated scalar γ_i is 0 in Q^*. This concludes our proof of Lemma 5. □

Proof (Proof of Claim 3). For $\mathcal{B}$ to have reached the Distinguishing Phase, $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_T}(Q^*)$ must be a valid GGM query; otherwise, $\mathcal{B}$ would have already ended Game 2 in the Extraction Phase. By definition of Q^{**}, the queries $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}(Q_1^{**})$ and $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}(Q_2^{**})$ must also be valid GGM queries. Knowing this, we want to show that $\mathcal{B}$ will be able to reconstruct these two queries 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 we demonstrated while proving Lemma 4, the expressions outputted by $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$ take a simple form that is guaranteed to be computable by $\mathcal{B}$ in Game 2. Thus, $\hat{D}$ can be computed successfully as a $\mathbb{G}_2$ element.

$D \in \mathbb{G}_1$, 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 $M_{2,i}^{(b)}$ for $i \in [5]$ and $b \in \{0,1\}$, $\mathcal{B}$ can't compute it as an element of $\mathbb{G}_1$, since these terms are not given by the $\mathsf{PK\text{-}CLH}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ challenger in Game 2. However, by

construction of Q^{**}, the scalars $\tau_{1,i}$, $\mu_{1,i}$, $\tau_{2,i}$, and $\mu_{2,i}$ are 0 for all $i \in [5]$. Thus, the expression for $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}(Q_1^{**})$ doesn't contain any $T_{1,i}$, $M_{1,i}$, $T_{2,i}^{(b)}$, or $M_{2,i}^{(b)}$ terms.

2. If the expression contains a signature component $h^{(i)}$, $b_1^{(i)}$, $s_1^{(i)}$, $b_2^{(b,i)}$, or $s_2^{(b,i)}$ from a $\mathcal{O}^{\mathsf{Sign}}$ query which $\mathcal{B}$ 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, Lemma 9 tells us that the inputs to any $\mathcal{O}^{\mathsf{Sign}}$ query made by $\mathcal{A}$ is computable by $\mathcal{B}$, 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. For $\mathcal{B}$ to fail when making a $\mathcal{O}^H$ query, it must have failed to compute the input string. However, Lemma 5 tells us that any hash h_i that appears in Q^* must have the form $h_i = h^{(k)}$ for some $k \in [q_h]$, where $h^{(k)}$ is part of the output $\left(\sigma_1^{(i)} = (h^{(i)}, b_1^{(i)}, s_1^{(i)}), \sigma_2^{(b,i)} = (h^{(i)}, b_2^{(b,i)}, s_2^{(b,i)})\right)$ of a $\mathcal{O}^{\mathsf{Sign}}$ query. We've just shown that such a $\mathcal{O}^{\mathsf{Sign}}$ query can be replicated by $\mathcal{B}$ in Game 2, so $\mathcal{B}$ will have access to the value h_i.

Since $D \in \mathbb{G}_1$ and $\hat{D} \in \mathbb{G}_2$ are both guaranteed to be computable, $\mathcal{B}$ will be able to successfully compute $e(D, \hat{D}) \in \mathbb{G}_T$ in Game 2. This concludes our proof of Claim 3. □

Proof (Proof of Claim 4). The fact that the reduction didn't end in the Extraction Phase tells us that Q^* is such that the discrete log polynomial $p^*(\vec{\kappa})$ of $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_T}(Q^*)$ is identically zero in one hybrid and identically nonzero in the other.

As with the proof of Lemma 5, we let $p^{(*,b)}(\vec{\kappa}) = p_1^{(*,b)}(\vec{\kappa}) \cdot p_2^{(*,b)}(\vec{\kappa})$, where depending on whether $b = 0$ or $b = 1$, we have:

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

$$
\begin{aligned}
p_1^{(*,1)}(\vec{\kappa}) =& \alpha_1 + \sum_{i\in[5]} (\tau_{1,i}(\eta_1\rho_{1,i}) + \tau_{2,i}(\eta_2\psi\rho_{1,i}) + \mu_{1,i}(\eta_1\rho_{1,i}\mathsf{sk}_{1,i}) + \mu_{2,i}(\eta_2\psi\rho_{1,i}\omega\mathsf{sk}_{1,i})) \\
&+ \sum_{k\in[q_h]} \gamma_k(\eta_{2+k}) + \sum_{k\in[q_\sigma]} \Bigg(\beta_{1,k}\left(\eta^{(k)}\sum_{j\in[2]}\rho_j^{(k)}\mathsf{sk}_{1,3+j}\right) + \beta_{2,k}\left(\eta^{(k)}\sum_{j\in[2]}\rho_j^{(k)}\omega\mathsf{sk}_{1,3+j}\right) \\
&+\zeta_{1,k}\left(\eta^{(k)}\mathsf{sk}_{1,1} + \sum_{j\in[2]} p_{M_j^{(k)}}(\vec{\kappa})\mathsf{sk}_{1,1+j}\right) + \zeta_{2,k}\left(\eta^{(k)}\omega\mathsf{sk}_{1,1} + \sum_{j\in[2]} p_{M_j^{(k)}}(\vec{\kappa})\omega\mathsf{sk}_{1,1+j}\right)\Bigg) \\
p_2^{(*,0)}(\vec{\kappa}) =& \alpha_2 + \sum_{j\in[5]} (\nu_{1,j}(\mathsf{sk}_{1,j}) + \nu_{2,j}(\mathsf{sk}_{2,j})) \\
p_2^{(*,1)}(\vec{\kappa}) =& \alpha_2 + \sum_{j\in[5]} (\nu_{1,j}(\mathsf{sk}_{1,j}) + \nu_{2,j}(\omega\mathsf{sk}_{1,j}))
\end{aligned}
$$

Now, we want to show that setting $\tau_{1,i} = \tau_{2,i} = \mu_{1,i} = \mu_{2,i} = 0$ for $i \in [5]$ does not change the value of $p^{(*,b)}(\vec{\kappa})$. In doing so, we will show that the modified query Q^{**} and its partitioned scalars Q_1^{**} and Q_2^{**} behave as we want: for $D \in \mathbb{G}_1$ and $\hat{D} \in \mathbb{G}_2$ computed according to the expressions $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}(Q_1^{**})$ and $\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}(Q_2^{**})$, respectively, $e(D, \hat{D}) = e(P, \hat{P})$ if and only if $p^{(*,b)}(\vec{\kappa})$ in that hybrid.

Our goal is to show that setting the scalars $\tau_{1,i}$, $\tau_{2,i}$, $\mu_{1,i}$, or $\mu_{2,i}$ does not affect the resulting value of $p^{(*,b)}(\vec{\kappa})$. For the most part, we do this by showing that the scalars had to have been 0 to begin with.

Every term in $p^{(*,b)}(\vec{\kappa})$ with one of the scalars $\tau_{1,i}$, $\tau_{2,i}$, $\mu_{1,i}$, or $\mu_{2,i}$ for $i \in [5]$ contains one of either $\rho_{1,i}$ or $\rho_{2,i}$. Lemma 3 tells us that for all $k \in [q_\sigma]$, $\rho_j^{(k)}$ and $p_{M_j^{(k)}}(\vec{\kappa})$ are not functions of $\rho_{1,i}$ or $\rho_{2,i}$ for $i \in [5]$ (i.e. the messages given to the signing oracle are independent of $\rho_{i,j}$). So, by inspection of $p^{(*,b)}(\vec{\kappa})$, we see that no terms besides those with scalars $\tau_{1,i}$, $\tau_{2,i}$, $\mu_{1,i}$, and $\mu_{2,i}$ include the indeterminants $\rho_{1,i}$ or $\rho_{2,i}$. Thus, these terms cannot cancel out with any other terms except each other. This is true in both Hybrid 1 and Hybrid 2. In more formal terms, $p^{(*,b)}(\rho_{*,*}) = \sum \tau_{i,j}q_{i,j}(\kappa) + \sum \mu_{i,j}q'_{i,j}(\kappa)$ where $p^{(*,b)}(\rho_{i,j})$ is all the terms of $p^{(*,b)}$ that contain any $\rho_{*,*}$. We can see that every term of $p^{(*,b)}(\rho_{i,j})$ contains some $\tau_{i,j}$ or $\mu_{i,j}$.

If it was the case that $p^{(*,b)}(\rho_{i,j})$ could not be zero in one hybrid and non-zero in the other, then if a single one of $\tau_{1,i}$, $\tau_{2,i}$, $\mu_{1,i}$, or $\mu_{2,i}$ for some $i \in [5]$ was nonzero in Q^*, both $p^{(*,0)}(\vec{\kappa})$ and $p^{(*,1)}(\vec{\kappa})$ would be nonzero. This contradicts what we know about Q^*, so the scalars must instead be 0. Now that we've shown that these terms contain indeterminants that bar them from canceling out with any other terms in $p^{(*,b)}(\vec{\kappa})$, we see if they are able to cancel out with each other.

Below is an exhaustive list of every such term in Hybrid 1 and Hybrid 2.

	$\tau_{1,i}$	$\tau_{2,i}$	$\mu_{1,i}$	$\mu_{2,i}$
α_2	$\eta_1\rho_{1,i}$	$\eta_1\rho_{2,i}$	$\boxed{\eta_1\rho_{1,i}\mathsf{sk}_{1,i}}$	$\eta_1\rho_{2,i}\mathsf{sk}_{2,i}$
$\nu_{1,j}$	$\boxed{\eta_1\rho_{1,i}\mathsf{sk}_{1,j}}$	$\eta_1\rho_{2,i}\mathsf{sk}_{1,j}$	$\eta_1\rho_{1,i}\mathsf{sk}_{1,i}\mathsf{sk}_{1,j}$	$\eta_1\rho_{2,i}\mathsf{sk}_{2,i}\mathsf{sk}_{1,j}$
$\nu_{2,j}$	$\eta_1\rho_{1,i}\mathsf{sk}_{2,j}$	$\eta_1\rho_{2,i}\mathsf{sk}_{2,j}$	$\eta_1\rho_{1,i}\mathsf{sk}_{1,i}\mathsf{sk}_{2,j}$	$\eta_1\rho_{2,i}\mathsf{sk}_{2,i}\mathsf{sk}_{2,j}$

	$\tau_{1,i}$	$\tau_{2,i}$	$\mu_{1,i}$	$\mu_{2,i}$
α_2	$\eta_1\rho_{1,i}$	$\eta_2\psi\rho_{1,i}$	$\boxed{\eta_1\rho_{1,i}\mathsf{sk}_{1,i}}$	$\eta_2\psi\rho_{1,i}\omega\mathsf{sk}_{1,i}$
$\nu_{1,j}$	$\boxed{\eta_1\rho_{1,i}\mathsf{sk}_{1,j}}$	$\eta_2\psi\rho_{1,i}\mathsf{sk}_{1,j}$	$\eta_1\rho_{1,i}\mathsf{sk}_{1,i}\mathsf{sk}_{1,j}$	$\eta_2\psi\rho_{1,i}\omega\mathsf{sk}_{1,i}\mathsf{sk}_{1,j}$
$\nu_{2,j}$	$\eta_1\rho_{1,i}\omega\mathsf{sk}_{1,j}$	$\eta_2\psi\rho_{1,i}\omega\mathsf{sk}_{1,j}$	$\eta_1\rho_{1,i}\mathsf{sk}_{1,i}\omega\mathsf{sk}_{1,j}$	$\eta_2\psi\rho_{1,i}\omega^2\mathsf{sk}_{1,i}\mathsf{sk}_{1,j}$

By inspection, we can conclude that none of these terms cancel each other out, with one exception that is shown framed in boxes. For all terms that can't cancel out with any other term, we conclude that the associated scalar must be 0 in Q^*, so forcibly setting it to 0 in Q^{**} doesn't change anything.

The one exception is that the terms $\mu_{1,i_1}\alpha_2(\eta_1\rho_{1,i_1}\mathsf{sk}_{1,i_1})$ and $\tau_{1,i_2}\nu_{1,j}(\eta_1\rho_{1,i_2}\mathsf{sk}_{1,j})$ can cancel specifically if $\mu_{1,i_1} = \tau_{1,i_2}$ for $i_1 = i_2 = i$, and $\nu_{1,j} = -\alpha_2$ for $j = i$. In both Hybrid 1 and Hybrid 2, we get the sum

$$\begin{aligned}\mu_{1,i}\alpha_2(\eta_1\rho_{1,i}\mathsf{sk}_{1,i}) + \tau_{1,i}\nu_{1,i}(\eta_1\rho_{1,i}\mathsf{sk}_{1,i}) &= \mu_{1,i}\alpha_2(\eta_1\rho_{1,i}\mathsf{sk}_{1,i}) + (\mu_{1,i})(-\alpha_2)(\eta_1\rho_{1,i}\mathsf{sk}_{1,i}) \\ &= (\mu_{1,i}\alpha_2 - \mu_{1,i}\alpha_2)(\eta_1\rho_{1,i}\mathsf{sk}_{1,i}) \\ &= 0\end{aligned}$$

so we see that these two terms do indeed cancel. This is the case for any $i \in [5]$, as long as the conditions above are met. Thus, there *is* a way for $\tau_{1,i}$ and $\mu_{1,i}$ to be nonzero in Q^*, and so setting them to 0 in Q^{**} *does* change something. However, since the sum of these terms is 0 in both hybrids, they do not affect the value of $p^{(*,b)}(\vec{\kappa})$ in either hybrid. Therefore, setting $\tau_{1,i}$ and $\mu_{1,i}$ to 0 may change the values of these scalars, but has no impact on the value of the discrete log $p^{(*,b)}(\vec{\kappa})$ in Hybrid 1 or Hybrid 2. If it was the case that $p^{(*,b)}(\vec{\kappa}) = 0$ in one hybrid and $p^{(*,b)}(\vec{\kappa}) \neq 0$ in the other, it will still be so now.

We've shown that the modification of scalars in Q^{**} doesn't change the output of the distinguishing query in either hybrid. This concludes our proof of Claim 4. □

Conclusion of Theorem 7 We have proven that Claim 3 and Claim 4 are true. Therefore, we have proven the public key class-hiding of $\mathbf{\Pi}_{\mathsf{MBGLS}}$. □

E Unforgeability Proofs for $\mathbf{\Pi}_{\mathsf{TMS}}$ and $\mathbf{\Pi}_{\mathsf{MBGLS}}$

In this section, we present the complete proofs of Theorems 13 and 6, in that order. We begin by proving the unforgeability (under Definition 5) of both the plan and cross-scheme correct versions of our $\mathbf{\Pi}_{\mathsf{TMS}}$ construction in Figure 4. We then prove the unforgeability (under Definition 17) of the cross-scheme correct version of the $\mathbf{\Pi}_{\mathsf{MBGLS}}$ construction in Figure 3. Note that the security of the plain version has already been proved in [50]).

E.1 Proof of Theorem 13

Recall that we have presented two versions of the $\mathbf{\Pi}_{\mathsf{TMS}}$ construction in Figure 4: the cross-scheme correct version that includes additional $\boxed{\text{boxed}}$ elements, and the plain version that does not. The former is a strict generalization of the latter, so if we can prove that it is existentially unforgeable under chosen tagged message attack (EUF-CtMA), we are done. However, our proof of unforgeability for the cross-scheme correct version of $\mathbf{\Pi}_{\mathsf{TMS}}$ relies on the unforgeability of the cross-scheme correct $\mathbf{\Pi}_{\mathsf{MBGLS}}$ (Theorem 6), whose proof is highly non-trivial and requires a lot of setup. Therefore, just as in Appendix D.1, 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 6. 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 (Proof of Theorem 13 (Unforgeability of $\mathbf{\Pi}_{\mathsf{TMS}}$*)).* Assume for the sake of contradiction that there exists a PPT adversary $\mathcal{A}$ that wins the existentially unforgeable under chosen tagged message attack game $\mathsf{EUF\text{-}CtMA}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{TMS}}}$ in Figure 1 with non-negligible advantage when executing the $\mathbf{\Pi}_{\mathsf{TMS}}$ construction from Figure 4. We construct a reduction $\mathcal{B}$ with black-box access to $\mathcal{A}$ that breaks the security of the $\mathsf{EUF\text{-}CtMA}_{\mathcal{B}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ game. Note that we have two cases:

1. In the proof of security of the plain $\mathbf{\Pi}_{\mathsf{TMS}}$ version, $\mathcal{B}$ plays $\mathsf{EUF\text{-}CtMA}_{\mathcal{B}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ against the plain version of $\mathbf{\Pi}_{\mathsf{MBGLS}}$.
2. In the proof of security of the cross-scheme correct $\mathbf{\Pi}_{\mathsf{TMS}}$ version, $\mathcal{B}$ plays $\mathsf{EUF\text{-}CtMA}_{\mathcal{B}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ against the cross-scheme correct version of $\mathbf{\Pi}_{\mathsf{MBGLS}}$.

Setup Phase. $\mathcal{B}$ simulates the setup phase as follows:

– $\mathcal{B}$ receives the public parameters $\mathsf{pp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, P, \hat{P}, e, H)$ and the global public key $\mathsf{pk}_0 = (\vec{N}^{(\mathsf{pk}_0)}, \boxed{\vec{M}^{(\mathsf{pk}_0)}}, \boxed{\vec{T}^{(\mathsf{pk})}})$ from the challenger.
– $\mathcal{B}$ forwards pp to $\mathcal{A}$, pretending to be the challenger of the $\mathsf{EUF\text{-}CtMA}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{TMS}}}$ and receives in return a set of corrupted parties $\mathcal{C}$. Assume WLOG that $|\mathcal{C}| = t-1$.

(Simulating) Key Generation Phase. $\mathcal{B}$ simulates the key generation phase as follows:

– For each corrupted party P_i with index $i \in \mathcal{C}$, $\mathcal{B}$ generates the partial signing keys sk_i and partial public keys pk_i as follows:
  - Randomly samples signing keys $\mathsf{sk}_i = (x_i, y_{i,1}, y_{i,2}, z_{i,1}, z_{i,2}) \xleftarrow{\$} \left(\mathbb{Z}_p^{\times}\right)^5$.

-
- Compute the partial public keys $\mathsf{pk}_{1,i} = \left(\vec{N}_1^{(\mathsf{pk}_i)}, \boxed{\vec{M}_1^{(\mathsf{pk}_i)}}, \boxed{\vec{T}_1^{(\mathsf{pk})}}\right)$, where:

$$\vec{N}^{(\mathsf{pk}_i)} = \left(\hat{X}_i, \hat{Y}_{1,i}, \hat{Y}_{2,i}, \hat{Z}_{1,i}, \hat{Z}_{2,i}\right) = \left(\hat{P}^{\mathsf{sk}_{i,j}}\right)_{j\in[5]}$$
$$\vec{M}^{(\mathsf{pk}_i)} = \left((T_{j,1}^{(\mathsf{pk})})^{\mathsf{sk}_{i,j}}\right)_{j\in[5]}$$

  Note that $\vec{T}^{(\mathsf{pk})}$ is the same tag outputted by the challenger in the setup phase, and thus don't need to be recomputed.

– For each honest party P_k with index $k \in \mathcal{H} = [n] \setminus \mathcal{C}$, $\mathcal{B}$ proceeds as follows:
- For all $i \in \tilde{\mathcal{T}} = \mathcal{C} \cup \{0\}$, $\mathcal{B}$ computes Lagrange coefficients evaluated at point k:

$$\tilde{\lambda}_{ki} = L_i^{\tilde{\mathcal{T}}}(k) = \prod_{j\in\tilde{\mathcal{T}} j\neq i} \frac{j-k}{j-i} \tag{9}$$

- Taking the public keys of corrupted parties $\{\mathsf{pk}_i\}_{i\in\mathcal{C}}$ and the global public key pk_0, $\mathcal{B}$ computes $\mathsf{pk}_{1,i} = \left(\vec{N}_1^{(\mathsf{pk}_k)}, \boxed{\vec{M}_1^{(\mathsf{pk}_k)}}, \boxed{\vec{T}_1^{(\mathsf{pk})}}\right)$ for all $k \in \mathcal{H}$, where:

$$\begin{aligned}\vec{N}_k &= (\hat{X}_k, \hat{Y}_{k,1}, \hat{Y}_{k,2}, \hat{Z}_{k,1}, \hat{Z}_{k,2}) \\ &= \left(\hat{X}^{\tilde{\lambda}_{k0}} \prod_{i\in\mathcal{C}} \hat{X}_i^{\tilde{\lambda}_{k,i}}, \hat{Y}_1^{\tilde{\lambda}_{k,0}} \prod_{i\in\mathcal{C}} \hat{Y}_{1,i}^{\tilde{\lambda}_{k,i}}, \hat{Y}_2^{\tilde{\lambda}_{k,0}} \prod_{i\in\mathcal{C}} \hat{Y}_{2,i}^{\tilde{\lambda}_{k,i}}, \hat{Z}_1^{\tilde{\lambda}_{k,0}} \prod_{i\in\mathcal{C}} \hat{Z}_{1,i}^{\tilde{\lambda}_{k,i}}, \hat{Z}_2^{\tilde{\lambda}_{k,0}} \prod_{i\in\mathcal{C}} \hat{Z}_{2,i}^{\tilde{\lambda}_{k,i}}\right) \\ \boxed{\vec{M}_k} &= \left(M_{1,0}^{\tilde{\lambda}_{k,0}} \prod_{i\in\mathcal{C}} M_{1,i}^{\tilde{\lambda}_{k,i}}, M_{2,0}^{\tilde{\lambda}_{k,0}} \prod_{i\in\mathcal{C}} M_{2,i}^{\tilde{\lambda}_{k,i}}, M_{3,0}^{\tilde{\lambda}_{k,0}} \prod_{i\in\mathcal{C}} M_{3,i}^{\tilde{\lambda}_{k,i}}, M_{4,0}^{\tilde{\lambda}_{k,0}} \prod_{i\in\mathcal{C}} M_{4,i}^{\tilde{\lambda}_{k,i}}, M_{5,0}^{\tilde{\lambda}_{k,0}} \prod_{i\in\mathcal{C}} M_{5,i}^{\tilde{\lambda}_{k,i}}\right)\end{aligned}$$

$\mathcal{B}$ sends the global public key pk_0, the full list of partial public keys $\vec{\mathsf{pk}} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, and the set of corrupted signing keys $\{\mathsf{sk}_i\}_{i\in\mathcal{C}}$ to $\mathcal{A}$.

(Simulating) Signing Phase. When $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{PSign}}(\cdot)$ with $(k, \mathsf{aux}, \vec{T}, \tau, (\vec{M}, \vec{N}))$ for an honest party index $k \in \mathcal{H}$, $\mathcal{B}$ does the following:

– $\mathcal{B}$ queries to the challenger of $\mathsf{EUF\text{-}CtMA}_{\mathcal{B}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ for a $\mathbf{\Pi}_{\mathsf{MBGLS}}$ signature on $(\vec{T}, \tau, \mathsf{aux}, (\vec{M}, \vec{N}))$ and receives $\sigma_0 = (h, b_0, s_0)$.
– For all corrupted party P_i where $i \in \mathcal{C}$, since sk_i is known, $\mathcal{B}$ can directly computes the partial signatures $\sigma_i = (h, b_i, s_i) \leftarrow \mathsf{ParSign}(\mathsf{sk}_i, \tau, \mathsf{aux}, (\vec{M}, \vec{N}))$.
– For all $k \in \tilde{\mathcal{T}} = \mathcal{C} \cup \{0\}$, $\mathcal{B}$ computes Lagrange coefficients $\tilde{\lambda}_{ki}$ as in Equation (9).
– For all honest party P_k where $k \in \mathcal{H}$, $\mathcal{B}$ computes the partial signatures $\sigma_k = (h, b_k, s_k) = \left(h, b_0^{\tilde{\lambda}_{k,0}} \prod_{i\in\mathcal{C}} b_i^{\tilde{\lambda}_{k,i}}, s_0^{\tilde{\lambda}_{k0,}} \prod_{i\in\mathcal{C}} s_i^{\tilde{\lambda}_{k,i}}\right)$ and sends σ_k to $\mathcal{A}$.

Output Phase. At the end of the game, $\mathcal{A}$ outputs its forgery $(\vec{M}^*, \vec{N}^*, \vec{T}^*, \tau^*, \sigma^*)$, which has a a non-negligible chance of being a valid forgery. $\mathcal{B}$ forwards $(\vec{M}^*, \vec{N}^*, \vec{T}^*, \tau^*, \sigma^*)$ to the $\mathsf{EUF\text{-}CtMA}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ challenger, finishing the game.

Analysis. First observe that the public parameter pp that $\mathcal{B}$ receives from the $\mathsf{EUF\text{-}CtMA}_{\mathcal{B}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ challenger is similarly distributed to the public parameter that $\mathcal{A}$ would receive from the *real* challenger in the $\mathsf{EUF\text{-}CtMA}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{TMS}}}$ game. Similarly, in the simulated key generation phase, the global public key, the full set of partial public keys, and the set of corrupted signing keys that $\mathcal{B}$ generates are similarly distributed to the keys that $\mathcal{A}$ would receive from the real challenger in the $\mathsf{EUF\text{-}CtMA}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{TMS}}}$ game. Thus, $\mathcal{B}$ simulates $\mathsf{EUF\text{-}CtMA}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{TMS}}}$ exactly as in a real execution.

Now observe that, if $\mathcal{A}$'s outputs a winning forgery $(\vec{M}^*, \vec{N}^*, \vec{T}^*, \tau^*, \sigma^*)$, then $\mathcal{B}$ will also win its game against the $\mathsf{EUF\text{-}CtMA}_{\mathcal{B}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ challenger. By assumption, $\mathcal{A}$ has non-negligible advantage in $\mathsf{EUF\text{-}CtMA}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{TMS}}}$. Thus, $\mathcal{B}$ will also win its game with non-negligible advantage. This contradicts the security of $\mathsf{EUF\text{-}CtMA}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ (by Theorem 7), so our initial assumption must be false.

Therefore, $\mathbf{\Pi}_{\mathsf{TMS}}$ is public key class-hiding under Definition 8. $\square$

E.2 Proof of Theorem 6

Now we provide the proof of Theorem 6, which states the unforgeability (under Definition 5) of the cross-scheme correct version of the $\mathbf{\Pi}_{\mathsf{MBGLS}}$ construction in Figure 3.

Proof (of Theorem 6 (Unforgeability of $\mathbf{\Pi}_{\mathsf{MBGLS}}$*)).* We will prove Theorem 6 in the generic group model (GGM). In addition the the signing oracle $\mathcal{O}^{\mathsf{Sign}}$ and hash oracle $\mathcal{O}^H$, the adversary $\mathcal{A}$ will also have access to the GGM oracles $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$, $\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}$, and $\mathcal{O}_{\mathbb{G}_T}^{\mathsf{GGM}}$. We will define (or in some cases, redefine) these oracles below.

Redefining the Game. 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 $\vec{N} = \mathsf{pk} = (\hat{X}, \hat{Y}_1, \hat{Y}_2, \hat{Z}_1, \hat{Z}_2) = (\hat{P}^x, \hat{P}^{y_1}, \hat{P}^{y_2}, \hat{P}^{z_1}, \hat{P}^{z_2})$, $h = H(P^{\rho_1}||P^{\rho_2}||P^{\rho_3}||P^{\rho_4}||P^{\rho_5}||\vec{N})$, $\vec{T} = (T_i)_{i\in[5]} = (h^{\rho_i})_{i\in[5]}$, and $\vec{M} = (T_1^x, T_2^{y_1}, T_3^{y_2}, T_4^{z_1}, T_5^{z_2})$. $\mathcal{B}$ stores all these encodings and sends them to $\mathcal{A}$.

$\mathcal{A}$ 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 $\mathcal{O}^H : \{0,1\} \to \mathbb{G}_1$, 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 $(\vec{M}, \vec{N}, \tau)$ returns a signature σ.

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

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

Here n_h represents the number of queries of $\mathcal{O}^H$, and n_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 $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$. 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 reduction and proof, we will focus on the discrete logs of group elements returned by GGM oracles. These discrete logs are polynomials in $\mathbb{Z}_p^\times[\vec{\kappa}]$, with indeterminants

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

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 $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$ and $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$ as polynomials of these indeterminants:

$$\begin{aligned}&\alpha + \sum_{i\in[5]} (\mu_i(\eta\rho_i\mathsf{sk}_i) + \tau_i(\eta\rho_i)) + \sum_{i\in[n_h]} \gamma_i(\eta_i) + \sum_{k\in[n_\sigma]} \left(\beta_k \left(\sum_{j\in[2]} (p_{T_j^{(k)}}(\vec{\kappa})\mathsf{sk}_j\right)\right.\\ &\quad \left. + \zeta_k \left(\eta^{(k)}\mathsf{sk}_1 + \sum_{j\in[2]} p_{M_j^{(k)}}(\vec{\kappa})\mathsf{sk}_j\right)\right) \alpha + \sum_{i\in[5]} \nu_i(\mathsf{sk}_i)\end{aligned}$$

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 $\mathcal{A}$. 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 $i \in [n_\sigma]$).

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 GGM oracle, it returns its forgery $(\vec{T}^*, (\vec{M}^*, \vec{N}^*), \sigma^*)$, where $|\vec{T}^*| = |\tau^*| = |\vec{M}^*| = |\vec{N}^*| = 2$, 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:

$$e(h^*, \hat{X}) \prod_{j\in[2]} e(M_j^*, \hat{Y}_j) = e(s^*, \hat{P})$$
$$e(b^*, \hat{P}) = \prod_{j\in[2]} e(T_j^*, \hat{Z}_j)$$
$$\bigwedge_{j\in[2]} e(T_j^*, \hat{N}_j^*) = e(M_j^*, \hat{P})$$

If the conditions are passed successfully, $\mathcal{A}$ wins the game.

Reductions. Suppose that there exists an adversary $\mathcal{A}$ to win Game 1 with non-negligible probability. We construct a reduction $\mathcal{B}$ that will run $\mathcal{A}$ as a subroutine to win Game 2. $\mathcal{B}$ functions as follows:

1. **Challenge Phase.** $\mathcal{B}$ challenges $\mathcal{A}$ to Game 1. As per the definition, $\mathcal{A}$ makes queries to each of $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$, $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$, $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_T}$, $\mathcal{O}^{H}$, and $\mathcal{O}^{\mathsf{Sign}}$. For each valid query, $\mathcal{B}$ responds honestly with encodings generated according to the game's definition, and also keeps track of said query. $\mathcal{B}$ does *not* keep track of any invalid queries (e.g. a $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$ 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 thus have no use in creating a forgery. Once $\mathcal{A}$ has made all of its queries, it sends its forgery $(\vec{T}^*, (\vec{M}^*, \vec{N}^*), \sigma^*)$ as a set of encodings, which has a non-negligible probability of being valid when decoded.
2. **Query Phase.** $\mathcal{B}$ plays Game 2 against the $\mathsf{EUF\text{-}CtMA}_{\mathcal{A}}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}$ challenger, who begins by sending $\vec{\mathsf{pk}}$. $\mathcal{B}$ sets $\vec{N} = \vec{\mathsf{pk}}$. $\mathcal{B}$'s goal is now to make all the same queries that $\mathcal{A}$ made. However, since $\vec{T}$ and $\vec{M}$ are given in Game 1 but not 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 for use in future computations. (For efficiency, these values can be stored in a dictionary with GGM encodings as keys and group elements as values). Note that $\mathcal{B}$ will not be able to compute this query if it contains any T_i or M_i for some $i \in [5]$, or if it contains the output of some previous query that $\mathcal{B}$ did not successfully compute.
   - 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. (By inspection of the $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$ definition, we can see that if $\mathcal{A}$'s original query was valid, $\mathcal{B}$ is guaranteed to succeed in its computation).

  - If it is a $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_T}$ query, $\mathcal{B}$ does nothing. $\mathbb{G}_T$ elements are not part of a valid forgery or any valid input to any query $\mathcal{A}$ can make, so this is not a necessary computation.
  - If it is a $\mathcal{O}^{\mathsf{Sign}}$, 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 a $\mathcal{O}^{\mathsf{Sign}}$ query to the $\mathsf{EUF\text{-}CtMA}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}_{\mathcal{A}}$ 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 $\mathcal{B}$ will not be able to successfully make this $\mathcal{O}^{\mathsf{Sign}}$ query.
  - 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 two encodings for $\mathbb{G}_2$ elements (i.e. $G_1\|G_2\|\hat{G}_3\|\hat{G}_4$), then $\mathcal{B}$ tries to find the computed group element associated with each encoding. If successful, $\mathcal{B}$ sends the string of those four group elements as a $\mathcal{O}^{H}$ query with the $\mathsf{EUF\text{-}CtMA}^{\mathbf{\Pi}_{\mathsf{MBGLS}}}_{\mathcal{A}}$ 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 $\mathcal{B}$ will not be able to successfully make this $\mathcal{O}^{\mathsf{Sign}}$ query. Or, if the input to this query does *not* have the form $G_1\|G_2\|\hat{G}_3\|\hat{G}_4$, $\mathcal{B}$ 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 produce a forgery based on $\mathcal{A}$'s forgery).
3. **Forgery Phase.** $\mathcal{B}$'s goal is to compute a modified signature σ^{**} based on $\mathcal{A}$'s forged signature $\sigma^* = (h^*, b^*, s^*)$. First, $\mathcal{B}$ checks each GGM encoding of $\vec{T}^*$, $\vec{M}^*$, $\vec{N}^*$, and b^* to try to find its associated group element. If a single encoding corresponds to an unsuccessful query, $\mathcal{B}$ immediately outputs $\perp$ and loses Game 2.
   Then, $\mathcal{B}$ finds the $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$ query expressions for h^* and s^* and modifies them, setting the scalars τ_i and μ_i to 0 (thereby forcibly removing any T_i and M_i elements). We denote these modified expressions by h^{**} and s^{**}. $\mathcal{B}$ now tries computing the values of h^{**} and s^{**} as group elements. If either of these computations fails (i.e. there is a term in the expression that corresponds to some previous failed query), $\mathcal{B}$ outputs $\perp$ and loses Game 2. Otherwise, $\mathcal{B}$ sets $\sigma^{**} = (h^{**}, b^*, s^{**})$ and sends the resulting forgery $(\vec{T}^*, (\vec{M}^*, \vec{N}^*), \sigma^{**})$ to the challenger, which ends Game 2.

We make the following claims about this reduction:

Claim 5 *If $\mathcal{A}$ wins Game 1, then $\mathcal{B}$ will be able to successfully compute $\vec{T}^*$, $\vec{M}^*$, and $\vec{N}^*$, as (vectors of) group elements in the Forgery Phase.*

Claim 6 *If $\mathcal{A}$ wins Game 1, then $\mathcal{B}$ will be able to successfully compute h^{**}, b^*, and s^{**} as group elements in the Forgery Phase.*

Claim 7 *If $\mathcal{A}$ wins Game 1, then $\mathcal{B}$'s modified signature $\sigma^{**} = (h^{**}, b^*, s^{**})$ defined in the Forgery Phase is valid in the forgery $(\vec{T}^*, (\vec{M}^*, \vec{N}^*), \sigma^{**})$.*

If all three claims are correct (and we will prove that they are), then whenever $\mathcal{A}$ wins Game 1, $\mathcal{B}$ will successfully compute $(\vec{T}^*, (\vec{M}^*, \vec{N}^*), \sigma^{**})$, which will be a valid forgery. Thus, if $\mathcal{A}$ can win Game 1 with non-negligible probability, $\mathcal{B}$ can win Game 2 with non-negligible probability. This breaks the unforgeability of $\mathbf{\Pi}_{\mathsf{MBGLS}}$, which is a contradiction—so no adversary can exist to win Game 2 with non-negligible probability. Thus, if we're able to prove each of these three claims, we will have proven the unforgeability of the cross-scheme correct version of $\mathbf{\Pi}_{\mathsf{MBGLS}}$.

Proof of the Reduction. As with the public key class-hiding proof, some groundwork will need to be set before we can prove these three claims. Thankfully, much of this groundwork is the same. And because many of the lemmas we proved in Appendix D.2 are generalizations of the lemmas we need here (since they deal with two sets of keys, and here we only need one), we don't need to re-prove them.

We will begin by re-writing Lemmas 1, 2, 3, and 4, and Corollary 1 as Lemmas 6, 7, 8, and 9, and Corollary 2, respectively. We will then prove that the terms $\vec{T}^*$, $\vec{M}^*$, and $\vec{N}^*$ in $\mathcal{A}$'s winning forgery are independent of any ρ_i terms (Lemma 10) and any hashes except those outputted in a previous signature query (Lemma 11). All of this will allow us to prove that $\mathcal{B}$ can compute each expression in $\vec{T}^*$, $\vec{M}^*$, and $\vec{N}^*$ as a group element in Game 2 (Claim 5).

At this point the remaining two claims won't require nearly as much work to prove, as they'll rely on some of the same lemmas. We will prove that b^* is independent of any ρ_i terms (Corollary 3), and that σ^* is independent of any hashes except those outputted in a previous signature query (Lemma 12). This will allow us to prove that $\mathcal{B}$ can compute each expression in the modified signature σ^{**} as a group element in Game 2 (Claim 6). Finally, we will prove that this modified forgery is a valid forgery Claim 7.

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 $h^{(i)}$, $b^{(i)}$, 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. This lemma follows directly from Lemma 1. □

Lemma 7. *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 signature $\sigma^{(k)} = (h^{(k)}, b^{(k)}, s^{(k)})$ outputted by this $\mathcal{O}^{\mathsf{Sign}}$ query.*

Proof. This lemma follows directly from Lemma 2. □

Lemma 8. *If $(M^{(k)}, N^{(k)}, \tau^{(k)})$ is the input to a successful $\mathcal{O}^{\mathsf{Sign}}$ query made by $\mathcal{A}$, then the expressions composing $M^{(k)}$, $N^{(k)}$, and $\tau^{(k)}$ are not functions of ρ_i.*

Proof. This lemma follows directly from Lemma 3. □

Corollary 2. *If $(M^{(k)}, N^{(k)}, \tau^{(k)})$ is the input to a successful $\mathcal{O}^{\mathsf{Sign}}$ query made by $\mathcal{A}$, then the expressions composing the output $\sigma^{(k)} = (h^{(k)}, b^{(k)}, s^{(k)})$ are not functions of ρ_i for any $i \in [5]$.*

Proof. This corollary follows directly from Corollary 1. □

Lemma 9. *If $(M^{(k)}, N^{(k)}, \tau^{(k)})$ is the input to a successful $\mathcal{O}^{\mathsf{Sign}}$ query made by $\mathcal{A}$, then the expressions composing $M^{(k)}$, $N^{(k)}$, and $\tau^{(k)}$ can all be computed by $\mathcal{B}$ in Game 2.*

Proof. This lemma follows directly from Lemma 4. □

Lemma 10. *If $\mathcal{A}$ wins Game 1 with forgery $(\vec{T}^*, (\vec{M}^*, \vec{N}^*), \sigma^*)$, then the expressions composing $\vec{T}^*$, $\vec{M}^*$, and $\vec{N}^*$ are not functions of ρ_i for any $i \in [5]$.*

Proof. Suppose that for some $j \in [2]$, one of M_j^*, $\hat{N}_j^*$, or T_j^* is a function of some ρ_i for $i \in [5]$. We will examine each possibility one by one.

1. Suppose it is $\hat{N}_j^*$ that is a function of some ρ_i. Since $\hat{N}_j^* \in \mathbb{G}_2$, it must be the result of some $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$ query. However, by simply inspecting the definition of $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$, we see that it is impossible for any output of $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$ to include any ρ_i indeterminant. Thus, $\hat{N}_j^*$ can't be a function of ρ_i for any $i \in [5]$.
2. Suppose instead that M_j^* is a function of some ρ_i. Since $M_j^* \in \mathbb{G}_1$, it is the result of some $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$ query. Corollary 2 tells us that no output $\sigma^{(k)}$ of a $\mathcal{O}^{\mathsf{Sign}}$ query for any $k \in [q_\sigma]$ is a function of ρ_i. Likewise, any output h_k of a $\mathcal{O}^H$ query for any $k \in [q_h]$ is only a function of some new indeterminant η_k. Thus, by definition of $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$, the only way to get some ρ_i value in the output is to query with nonzero coefficients τ_i and μ_i for T_i and M_i, respectively. In other words, we can write the discrete log polynomial of M_j^* as

$$p_{M_j^*}(\vec{\kappa}) = \sum_{i\in[5]} \tau_i^{(j)}(\eta\rho_i) + \sum_{i\in[5]} \mu_i^{(j)}(\eta\rho_i \mathsf{sk}_i) + \cdots$$

where there may be other terms in the query. If $\mathcal{A}$'s forgery is valid, it must pass the three verification conditions. We can write the discrete log of the first verification condition, given what we now know about M_j^*, as the equation

$$p_{h^*}(\vec{\kappa})^* \mathsf{sk}_1 + \sum_{j\in[2]} \left(\sum_{i\in[5]} \tau_i^{(j)}(\eta\rho_i) + \sum_{i\in[5]} \mu_i^{(j)}(\eta\rho_i \mathsf{sk}_i) + \cdots \right) \mathsf{sk}_{1+j} = p_{s^*}(\vec{\kappa}),$$

where $p_{h^*}(\vec{\kappa})$ and $p_{s^*}(\vec{\kappa})$ are the discrete log polynomials of h^* and s^*, respectively. We can conclude by inspection that the terms on the LHS of this equation all contain different indeterminants, so there is no choice of scalars that can cancel any of these terms out. Thus, any term that is nonzero in $p_{M_j^*}\vec{\kappa}$ (and thus the LHS) must also be nonzero in the RHS, in the polynomial $p_{s^*}(\vec{\kappa})$. We will use this to narrow down the form that $p_{M_j^*}(\vec{\kappa})$ can take.

If any single $\mu_i^{(j)}$ is nonzero, there would be a $\mathsf{sk}_i\mathsf{sk}_{1+j}$ term on the LHS. However, by definition of $\mathcal{O}^{\mathsf{GGM}}$, there is no $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$ query that could return s^* such that its discrete log $p_{s^*}(\vec{\kappa})$ contains $\mathsf{sk}_i\mathsf{sk}_{1+j}$. Thus, to satisfy Equation 1, we conclude that all $\mu_i^{(j)} = 0$. Applying similar logic to the $\tau_i^{(j)}$ scalars, any such nonzero scalar will result in a $\eta\rho_i\mathsf{sk}_{1+j}$ term. These terms can be queried as multiples of M_{1+j}, but only if $i = 1+j$. Thus, the only $\tau_i^{(j)}$ values that may be nonzero are $\tau_2^{(1)}$ and $\tau_3^{(2)}$. Using all this, we can rewrite our definition of the discrete log of M_j^*:

$$p_{M_j^*}(\vec{\kappa}) = \tau_i^{(j)}(\eta\rho_{1+j}) + \cdots$$

With this in mind, we can write the discrete log of the third verification condition as the equation

$$p_{T_j^*}(\vec{\kappa})p_{N_j^*}(\vec{\kappa}) = \tau_i^{(j)}(\eta\rho_{1+j}) + \cdots \tag{10}$$

for $j \in [2]$ where $p_{T_j^*}(\vec{\kappa})$ and $p_{N_j^*}(\vec{\kappa})$ are the discrete log polynomials of T_j^* and N_j^*, respectively. The definition of $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$ tells us that $p_{N_j^*}(\vec{\kappa})$ can contain neither η nor ρ_{1+j}. So, to satisfy the above equation, $p_{T_j^*}(\vec{\kappa})$ must be of the form

$$p_{T_j^*}(\vec{\kappa}) = c^{(j)}(\eta\rho_{1+j}) + \cdots ,$$

for some constant $c^{(j)}$. Knowing this, we can write the discrete log of the second verification condition as

$$p_{b^*}(\vec{\kappa}) = \Big(c^{(1)}(\eta\rho_2) + \cdots\Big)\mathsf{sk}_4 + \Big(c^{(1)}(\eta\rho_3) + \cdots\Big)\mathsf{sk}_5,$$

where $p_{b^*}(\vec{\kappa})$ is the discrete log polynomial of b^*. However, the RHS contains terms $\rho_2\mathsf{sk}_4$ and $\rho_3\mathsf{sk}_5$, which are not possible to get from a query of $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$. Therefore, there is no way to query a valid b^* unless both $c^{(j)} = 0$, and thereby $\tau_i^{(j)} = 0$. Thus, if $\mathcal{A}$'s forgery is valid, $\vec{M}^*$ must not be a function of ρ_i for any $i \in [5]$.

3. Finally, suppose that T_j^* is a function of some ρ_i. The discrete log equation of the first verification condition is

$$p_{T_j^*}(\vec{\kappa})p_{N_j^*}(\vec{\kappa}) = p_{M_j^*}(\vec{\kappa}) \tag{11}$$

However, we have concluded that neither $\hat{N}_j^*$ nor M_j^* are functions of any ρ_i, so neither $p_{N_j^*}(\vec{\kappa})$ nor $p_{M_j^*}(\vec{\kappa})$ contain any terms with ρ_i. For the equation above to be satisfied, it must be the case that $p_{T_j^*}(\vec{\kappa})$ also doesn't contain any ρ_i terms, so T_j^* is independent of ρ_i for any $i \in [5]$.

We've shown that if $(\vec{M}^*, \vec{N}^*, \vec{T}^*)$ is a valid forgery produced by $\mathcal{A}$, then for any $j \in [2]$, neither M_j^* nor $\hat{N}_j^*$ nor T_j^* is a function of ρ_i for any $i \in [5]$. This concludes our proof of Lemma 10. □

Lemma 11. *If $\mathcal{A}$ wins Game 1 with forgery $(\vec{T}^*, (\vec{M}^*, \vec{N}^*), \sigma^*)$, then the expressions composing $\vec{T}^*$, $\vec{M}^*$, and $\vec{N}^*$ are not functions 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 some signature $\sigma^{(k)} = (h^{(k)}, b^{(k)}, s^{(k)})$ outputted by $\mathcal{O}^{\mathsf{Sign}}$.*

Proof. Suppose that for some $j \in [2]$, one of M_j^*, $\hat{N}_j^*$, or T_j^* is a function of some hash h_i. We can immediately conclude that it isn't $\hat{N}_j^*$, since $\hat{N}_j^* \in \mathbb{G}_2$ and $h_i \in \mathbb{G}_1$. So, either M_j^* or T_j^* is a function of h_i.

If $\mathcal{A}$'s forgery is valid, it must pass all three verification conditions. The third condition, $\bigwedge_{j\in[2]} e(T_j^*, N_j^*) = e(M_j^*, \hat{P})$, has discrete log

$$\bigwedge_{j\in[2]} p_{T_j^*}(\vec{\kappa}) p_{N_j^*}(\vec{\kappa}) = p_{M_j^*}(\vec{\kappa}) \tag{12}$$

where $p_{T_j^*}(\vec{\kappa}), p_{N_j^*}(\vec{\kappa}), p_{M_j^*}(\vec{\kappa}) \in \mathbb{Z}_p^\times[\vec{\kappa}]$ are the discrete logs of T_j^*, N_j^*, and M_j^*, respectively. By definition of $\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}$, $p_{N_j^*}(\vec{\kappa})$ can't contain the indeterminant η_i (where η_i is the discrete log of the hash h_i). Thus, if either $p_{T_j^*}(\vec{\kappa})$ or $p_{M_j^*}(\vec{\kappa})$ contains η_i (and by our initial assumption, one of them does), then so must the other, to satisfy the equation above.

If $\mathcal{A}$'s forgery is valid, it must also pass the second verification condition, $e(b^*, \hat{P}) = \prod_{j\in[2]} e(T_j^*, \hat{Z}_j)$, where b^* is part of the forged signature $\sigma^* = (h^*, b^*, s^*)$. The discrete log of this condition is the equation

$$p_{b^*}(\vec{\kappa}) = p_{T_j^*}(\vec{\kappa})\mathsf{sk}_{3+j} \tag{13}$$

where $p_{b^*}(\vec{\kappa}) \in \mathbb{Z}_p^\times[\vec{\kappa}]$ is the discrete log of b^*. Knowing that $p_{T_j^*}(\vec{\kappa})$ contains the indeterminant η_i lets us rewrite the above equation as

$$p_{b^*}(\vec{\kappa}) = (\gamma_i \eta_i + \cdots)\mathsf{sk}_{3+j} \tag{14}$$

where γ_i is some scalar in $\mathbb{Z}_p^\times$. To satisfy this equation, $p_{b^*}(\vec{\kappa})$ must contain a $\eta_i \mathsf{sk}_{3+j}$ term. By definition of $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$, the only possible way to get this is by using the $b^{(k)}$ term of the signature $\sigma^{(k)} = (h^{(k)}, b^{(k)}, s^{(k)})$ outputted by $\mathcal{O}^{\mathsf{Sign}}$ for some value $k \in [q_\sigma]$. $b^{(k)}$ has discrete log

$$\log_P b^{(k)} = \eta^{(k)} \rho_1^{(k)} \mathsf{sk}_4 + \eta^{(k)} \rho_2^{(k)} \mathsf{sk}_5$$

where $\eta^{(k)}$ is the discrete log of $h^{(k)}$ and $\rho_1^{(k)}, \rho_2^{(k)} \in \mathbb{Z}_p^\times$. The only way to have a $\eta_i \mathsf{sk}_{3+j}$ term as part of $b^{(k)}$ is to set $\eta_i = \eta^{(k)}$. In other words, for the second verification equation to be satisfied, if T_j^* or M_j^* is a function of some hash h_i, that hash must be $h_i = h^{(k)}$. This concludes our proof of Lemma 11. □

Now we have finally laid the necessary groundwork to prove Claim 5.

Proof (of Claim 5). In the GGM, each of $\vec{T}^*$, $\vec{M}^*$, and $\vec{N}^*$ is a vector of encodings resulting from $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$ and $\mathcal{O}_{\mathbb{G}_2}^{\mathsf{GGM}}$ queries. We know these to be valid queries, because we've assumed that $\mathcal{A}$ has won Game 1 with the forgery $(\vec{T}^*, (\vec{M}^*, \vec{N}^*), \sigma^*)$.

Now it is a matter of showing that $\mathcal{B}$ can successfully compute each encoding's expression as a $\mathbb{G}_1$ or $\mathbb{G}_2$ element.

$\vec{N}^*$ is a vector of encodings outputted by $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$. As we demonstrated while proving Lemma 9, the expressions outputted by $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_2}$ take a simple form that is guaranteed to be computable by $\mathcal{B}$ in Game 2. Thus, $\vec{N}^*$ can be computed successfully as a $\mathbb{G}_2$ element.

$\vec{T}^*$ and $\vec{M}^*$, on the other hand, are vectors of encodings outputted by $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$. 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_i or M_i for $i \in [5]$, $\mathcal{B}$ can't compute it as an element of $\mathbb{G}_1$, since T_i and M_i are not given to it in Game 2. However, Lemma 10 tells us that no expression in $\vec{T}^*$ and $\vec{M}^*$ is a function of ρ_i for any $i \in [5]$. Since each T_i and M_i is a function of ρ_i, then no $\vec{T}^*$ or $\vec{M}^*$ expression can contain any T_i or M_i, so this is not the case.
2. If the expression contains a signature component $h^{(k)}$, $b^{(k)}$, or $s^{(k)}$ from a $\mathcal{O}^{\mathsf{Sign}}$ query which $\mathcal{B}$ 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, Lemma 9 tells us that the inputs to any $\mathcal{O}^{\mathsf{Sign}}$ query made by $\mathcal{A}$ is computable by $\mathcal{B}$, 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. For $\mathcal{B}$ to fail when making a $\mathcal{O}^H$ query, it must have failed to compute the input string. However, Lemma 11 tells us that hash h_i that appears in $\vec{T}^*$ or $\vec{M}^*$ must have the form $h_i = h^{(k)}$ for some $k \in [q_h]$, where $h^{(k)}$ is part of some previously-queried signature $\sigma^{(k)} = (h^{(k)}, b^{(k)}, s^{(k)})$. We've just shown that such a $\mathcal{O}^{\mathsf{Sign}}$ query can be replicated by $\mathcal{B}$ in Game 2, so $\mathcal{B}$ will have access to the value h_i.

Therefore, each expression composing $\vec{T}^*$, $\vec{M}^*$, and $\vec{N}^*$ can be computed by $\mathcal{B}$ in Game 2. This concludes our proof of Claim 5. □

Corollary 3. *If $\mathcal{A}$ wins Game 1 with forgery $(\vec{T}^*, (\vec{M}^*, \vec{N}^*), \sigma^* = (h^*, b^*, s^*))$, then the expression for b^* is not functions of ρ_i for any $i \in [5]$.*

Proof. This corollary follows directly from Lemma 10. Since $\mathcal{A}$'s forgery is valid, it must satisfy all three verification conditions. Take the second condition, whose discrete log is the equation

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

where $p_{b^*}(\vec{\kappa})$ and $p_{T_j^*}(\vec{\kappa})$ are the discrete log polynomials of b^* and T_j^*, respectively. Lemma 8 tells us that T_j^* (and by extension $p_{T_j^*}(\vec{\kappa})$) is independent of ρ_i for any $i \in [5]$. Thus, the RHS of the equation above does not contain any ρ_i

terms. Since this is a valid forgery, said equation must be satisfied, and so the LHS must also not contain any ρ_i terms. Therefore, $p_{b^*}(\vec{\kappa})$ (and by extension b^*) must not be a function of ρ_i for any $i \in [5]$. This concludes our proof of Corollary 3. $\square$

Lemma 12. *If $\mathcal{A}$ wins Game 1 with forgery $(\vec{T}^*, (\vec{M}^*, \vec{N}^*), \sigma^*)$, then the expressions composing $\sigma^* = (h^*, b^*, s^*)$ are not functions 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 some signature $\sigma^{(k)} = (h^{(k)}, b^{(k)}, s^{(k)})$ outputted by $\mathcal{O}^{\mathsf{Sign}}$.*

Proof. We will show one by one that neither b^* nor h^* nor s^* contain in its expression a nonzero h_i that isn't of the desired form. This proof relies on the fact that $\mathcal{A}$'s forgery is valid, so it must pass each of the three verification conditions.

1. Suppose that s^* is a function of some hash h_i. Since s^* is the result of some $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$ query, we can write its discrete log polynomial as $p_{s^*}(\vec{\kappa}) = \gamma_i \eta_i + \cdots$, where γ_i is a scalar in $\mathbb{Z}_p^\times$ and η_i is an indeterminant representing the discrete log of h_i. Using what we now know about s^*, we can write the discrete log of the first verification equation as
$$p_{h^*}(\vec{\kappa})\mathsf{sk}_1 + \sum_{j\in[2]} p_{M_j^*}(\vec{\kappa})\mathsf{sk}_{1+j} = \gamma_i\eta_i + \cdots \tag{15}$$
where $p_{h^*}(\vec{\kappa})$ and $p_{M_j^*}(\vec{\kappa})$ are the discrete log polynomials of h^* and M_j^*, respectively. We can see that all terms on the LHS contain either sk_1, sk_2, or sk_3, whereas $\gamma_i\eta_i$ contains none of these indeterminants. Thus, the verification equation can't be satisfied, which is a contradiction (as $\mathcal{A}$ is assumed to have produced a valid forgery). It must be the case that s^* isn't a function of any hash at all.
2. Suppose b^* is a function of some hash h_i. As before, we write its discrete log polynomial as $p_{b^*}(\vec{\kappa}) = \gamma_i\eta_i + \cdots$, where γ_i is a scalar in $\mathbb{Z}_p^\times$. Using this, we can write the discrete log of the second verification equation as
$$\gamma_i\eta_i + \cdots = \sum_{j\in[2]} p_{T_j^*}(\vec{\kappa})\mathsf{sk}_{3+j} \tag{16}$$
where $p_{T_j^*}(\vec{\kappa})$ is the discrete log of T_j^*. It is clear by inspection that the only way to satisfy this equation is to have $p_{T_j^*}(\vec{\kappa})$ contain η_i for some $j \in [2]$; that is, T_j^* must be a function of h_i. Lemma 11 tells us that, if this is the case, then $h_i = h^{(k)}$, where $h^{(k)}$ is part of the result of a $\mathcal{O}^{\mathsf{Sign}}$ query. Thus, if b^* is a function of some hash h_i, that hash must be of the desired form.
3. Finally, suppose that h^* is a function of some hash h_i. We write the discrete log polynomial of h^* as $p_{h^*}(\vec{\kappa}) = \gamma_i\eta_i + \cdots$ for some $\gamma_i \in \mathbb{Z}_p^\times$. We use this to write the discrete log of the first verification equation as
$$(\gamma_i\eta_i + \cdots)\mathsf{sk}_1 + \sum_{j\in[2]} p_{M_j^*}(\vec{\kappa})\mathsf{sk}_{1+j} = p_{s^*}(\vec{\kappa}) \tag{17}$$

where $p_{M_j^*}(\vec{\kappa})$ and $p_{s^*}(\vec{\kappa})$ are the discrete log polynomials of M_j^* and s^*, respectively. This condition can only be satisfied if $p_{s^*}(\vec{\kappa}) = \gamma_i \eta_i \mathsf{sk}_1 + \cdots$. Since $p_{s^*}(\vec{\kappa})$ is the result of a $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$ query, the only way to get $p_{s^*}(\vec{\kappa})$ in the desired form is to make use of some $s^{(k)}$, as part of the signature $\sigma^{(k)} = (h^{(k)}, b^{(k)}, s^{(k)})$ outputted by $\mathcal{O}^{\mathsf{Sign}}$ for some kth query. In other words,

$$\gamma_i \eta_i \mathsf{sk}_1 + \cdots = p_{s^*}(\vec{\kappa}) = \gamma_i s^{(k)} = \gamma_i(\eta^{(k)} \mathsf{sk}_1 + \cdots) + \cdots \tag{18}$$

where $\eta^{(k)}$ is the discrete log of $h^{(k)}$. For this to be satisfied, we would need $\eta_i = \eta^{(k)}$, which is what we want.

In all three cases, the respective component of $\sigma^* = (h^*, b^*, s^*)$ cannot be a function of any hash h_i except $h_i = h^{(k)}$ where $h^{(k)}$ results from a $\mathcal{O}^{\mathsf{Sign}}$ query for some $k \in [q_\sigma]$. This concludes our proof of Lemma 12. $\square$

Proof (of Claim 6). In the GGM, h^*, b^*, and s^* are encodings resulting from $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$ queries. We know these to be valid queries, because we've assumed that $\mathcal{A}$ has won Game 1 with the forgery $(\vec{T}^*, (\vec{M}^*, \vec{N}^*), \sigma^* = (h^*, b^*, s^*))$. The expressions for h^{**} and s^{**} can also be characterized as a $\mathcal{O}^{\mathsf{GGM}}_{\mathbb{G}_1}$ expression, just with the condition that the scalars τ_i and μ_i for all $i \in [5]$ are 0. 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_i or M_i for $i \in [5]$, $\mathcal{B}$ can't compute it as an element of $\mathbb{G}_1$, since T_i and M_i are not given to it in Game 2. However, the modified terms h^{**} and s^{**} are defined exactly so that they don't contain any T_i or M_i terms. Moreover, Corollary 3 tells us that b^* is independent of any ρ_i terms for $i \in [5]$. Since each T_i and M_i contains ρ_i, b^* must be independent of all T_i and M_i. Thus, neither of the three expressions for h^{**}, b^*, or s^{**} contains T_i or M_i for $i \in [5]$.
2. If the expression contains a signature component $h^{(k)}$, $b^{(k)}$, or $s^{(k)}$ from a $\mathcal{O}^{\mathsf{Sign}}$ query which $\mathcal{B}$ 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, Lemma 9 tells us that the inputs to any $\mathcal{O}^{\mathsf{Sign}}$ query made by $\mathcal{A}$ is computable by $\mathcal{B}$, 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. For $\mathcal{B}$ to fail when making a $\mathcal{O}^H$ query, it must have failed to compute the input string. However, Lemma 12 tells us that any hash h_i that appears in $\vec{T}^*$ or $\vec{M}^*$ must have the form $h_i = h^{(k)}$ for some $k \in [q_h]$, where $h^{(k)}$ is part of some previously-queried signature $\sigma^{(k)} = (h^{(k)}, b^{(k)}, s^{(k)})$. We've just shown that such a $\mathcal{O}^{\mathsf{Sign}}$ query can be replicated by $\mathcal{B}$ in Game 2, so $\mathcal{B}$ will have access to the value h_i.

Therefore, the expressions for h^{**}, b^*, and s^{**} can all be computed by $\mathcal{B}$ in Game 2. This concludes our proof of Claim 6. $\square$

Proof (of Claim 7). If $\mathcal{A}$'s forgery $(\vec{T}^*, (\vec{M}^*, \vec{N}^*), \sigma^* = (h^*, b^*, s^*)$ is valid, then it passes all three verification conditions. We want to show that $\mathcal{B}$'s modified forgery $(\vec{T}^*, (\vec{M}^*, \vec{N}^*), \sigma^{**} = (h^{**}, b^*, s^{**})$ also passes all three conditions. As a reminder, these three conditions are

$$e(h^{**}, \hat{X}) \prod_{j \in [2]} e(M_j^*, \hat{Y}_j) = e(s^{**}, \hat{P})$$
$$e(b^*, \hat{P}) = \prod_{j \in [2]} e(T_j^*, \hat{Z}_j)$$
$$\bigwedge_{j \in [2]} e(T_j^*, \hat{N}_j^*) = e(M_j^*, \hat{P})$$

The second and third check don't make use of the modified h^{**} and s^{**} terms. Since $\mathcal{A}$'s forgery passes these conditions, so does $\mathcal{B}$'s forgery. Thus, all that remains is to show that $\mathcal{B}$'s forgery passes the first verification check.

We know that h^* and s^* are both elements of $\mathbb{G}_1$, so they must be the results of some $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$ queries made by $\mathcal{A}$. We know the form that the discrete logs polynomials $p_{h^*}(\vec{\kappa})$ and $p_{s^*}(\vec{\kappa})$ or h^* and s^* must take, given by the definition of $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$. We are specifically interested in the T_i and M_i terms for $i \in [5]$, so we write

$$p_{h^*}(\vec{\kappa}) = \sum_{i \in [5]} \left(\tau_i^{(h^*)}(\eta\rho_i) + \mu_i^{(h^*)}(\eta\rho_i \mathsf{sk}_i) \right) + p'_{h^*}(\vec{\kappa})$$
$$p_{s^*}(\vec{\kappa}) = \sum_{i \in [5]} \left(\tau_i^{(s^*)}(\eta\rho_i) + \mu_i^{(s^*)}(\eta\rho_i \mathsf{sk}_i) \right) + p'_{s^*}(\vec{\kappa})$$

Here we define $p'_{h^*}(\vec{\kappa})$ and $p'_{s^*}(\vec{\kappa})$ as the "rest" of each respective polynomial, as defined by $\mathcal{O}_{\mathbb{G}_1}^{\mathsf{GGM}}$. (Before, we've used "$\cdots$" to denote this, but now we want to be more explicit). Using this, we can write the discrete log equation of the first verification condition:

$$\begin{aligned} &\left(\sum_{i \in [5]} \left(\tau_i^{(h^*)}(\eta\rho_i) + \mu_i^{(h^*)}(\eta\rho_i \mathsf{sk}_i) \right) + p'_{h^*}(\vec{\kappa}) \right) \mathsf{sk}_1 \\ &+ \sum_{j \in [2]} p_{M_j^*}(\vec{\kappa}) \mathsf{sk}_{1+j} = \sum_{i \in [5]} \left(\tau_i^{(s^*)}(\eta\rho_i) + \mu_i^{(s^*)}(\eta\rho_i \mathsf{sk}_i) \right) + p'_{s^*}(\vec{\kappa}) \end{aligned} \tag{19}$$

Lemma 10 tells us that $p_{M_j^*}(\vec{\kappa})$ is independent of $\{\rho_i\}_{i \in [5]}$, so the ρ_i terms written above are exhaustive. We can see by inspection that no choice of scalars can cancel out ρ_i terms on either side of the equation, so any term with ρ_i present on the LHS must also be present on the RHS. Our assumption that $\mathcal{A}$'s forgery is correct, and thus passes this verification check, allows us to make the following conclusions about said forgery:

- $\mu_i^{(h^*)} = 0$ for all $i \in [5]$. Otherwise, if any $\mu_i^{(h^*)}$ were nonzero, there would be a $\eta\rho_i \mathsf{sk}_i \mathsf{sk}_1$ term on the LHS. There is no way to query s^* such that $\eta\rho_i \mathsf{sk}_i \mathsf{sk}_1$ appears on the RHS, so the equation can only be satisfied if $\mu_i^{(h^*)} = 0$.

- $\tau_i^{(s^*)} = 0$ for all $i \in [5]$. Otherwise, if any $\tau_i^{(s^*)}$ were nonzero, there would be a $\eta\rho_i$ term on the RHS. There is no way to query s^* such that only $\eta\rho_i$ appears on the LHS; indeed, any term on the LHS containing ρ_i also contains sk_i. Thus, the equation can only be satisfied if $\tau_i^{(s^*)} = 0$.
- $\tau_i^{(h^*)} = 0$ for $1 \neq i \in [5]$. If any $\tau_i^{(h^*)}$ were nonzero, there would be a $\eta\rho_i\mathsf{sk}_1$ term on the LHS. s^* can be queried such that this $\eta\rho_i\mathsf{sk}_1$ term appears on the RHS, satisfying the equation, only if $i = 1$. For any $1 \neq i \in [5]$, the term $\eta\rho_i\mathsf{sk}_1$ could not appear on the RHS, so we must have $\tau_i^{(h^*)} = 0$.
- $\mu_i^{(s^*)} = 0$ for all $1 \neq i \in [5]$. A similar logic applies here. Since every term on the LHS contains sk_i, if this equation is to be satisfied, the only $\eta\rho_i\mathsf{sk}_i$ term that may appear on the RHS is $\eta\rho_1\mathsf{sk}_1$. This corresponds to $\mu_1^{(s^*)}$. For all other $1 \neq i \in [5]$, $\mu_i^{(s^*)} = 0$.

We can again rewrite the above equation with these facts in mind, omitting any terms with scalar coefficients of 0:

$$\left(\tau_1^{(h^*)}(\eta\rho_1) + p'_{h^*}(\vec{\kappa})\right)\mathsf{sk}_1 + \sum_{j\in[2]} p_{M_j^*}(\vec{\kappa})\mathsf{sk}_{1+j} = \mu_1^{(s^*)}(\eta\rho_1\mathsf{sk}_1) + p'_{s^*}(\vec{\kappa})$$

Since the terms $\tau_1^{(h^*)}(\eta\rho_1)$ and $\mu_1^{(s^*)}(\eta\rho_1\mathsf{sk}_1)$ are the only terms in this equation with ρ_1, and given our assumption that $\mathcal{A}$'s forgery passes all checks, we can conclude that $\tau_1^{(h^*)}(\eta\rho_1)\mathsf{sk}_1 = \mu_1^{(s^*)}(\eta\rho_1\mathsf{sk}_1)$. More specifically, we know that $\tau_1^{(h^*)} = \mu_1^{(s^*)}$. Algebraically, we can subtract this term from both sides to get

$$p'_{h^*}(\vec{\kappa})\mathsf{sk}_1 + \sum_{j\in[2]} p_{M_j^*}(\vec{\kappa})\mathsf{sk}_{1+j} = p'_{s^*}(\vec{\kappa}) \tag{20}$$

Notice that this is the discrete log of the first verification condition, where we've removed all $\{T_i\}_{i\in[5]}$ and $\{M_i\}_{i\in[5]}$ terms from h^* and s^*. In other words, it is exactly the discrete log of the first verification equation, applied on $\mathcal{B}$'s modified h^{**} and s^{**}. We can rewrite the equation above as

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

where $p_{h^{**}}(\vec{\kappa})$ and $p_{s^{**}}(\vec{\kappa})$ are the discrete logs of h^{**} and s^{**}, respectively. Therefore, if $\mathcal{A}$'s forgery passes all three verification checks, $\mathcal{B}$'s modified forgery will too. This concludes our proof of Claim 7. □

Conclusion of Theorem 6 We have proven that Claim 5, Claim 6, and Claim 7 are true. Therefore, we have proven the unforgeability of $\mathbf{\Pi}_{\mathsf{MBGLS}}$.