Non-Interactive Threshold Mercurial Signatures with Applications to Threshold DAC

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 [Cha85], first realized by Camenisch and Lysyanskaya [CL04], and then improved and extended in many subsequent works [CKL$^+$16, CDHK15]. Anonymous credentials allow a user to prove in zero-knowledge that they know a signature without supplying it in the clear. They are, seemingly, the perfect answer to legislative efforts towards privacy-preserving digital identity, such as the EU's Digital Identity Wallet regulation (EUDIW) [eID24, BBH$^+$24]. 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 [CL06]. 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 [CL19] 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 [CL21, CLPK22, MBG+23, PM23, ANPKT24, GLM+24b].

Threshold signatures [Des88, DF90] 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 [SAB+19, DKL+23]. Thresholdizing a mercurial signatures can strengthen both anonymity and unforgeability [ANPKT24]. 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 [ANPKT24] and used to construct mix-nets in [ANO+25]. 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. [MBG+23] which is itself based on the structure-preserving signature of Ghadafi [Gha16].
- We further extend the construction in [MBG+23] 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 [MSM24], 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 [CL01]. In general, non-transferability is achieved by signing user public keys, enabling users to prove their identity. Since [MSM24] 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) [Sho97]. Constructing mercurial signatures is known to at least require non-interactive assumptions [BFR24], and many constructions use generic or algebraic group models [MBG+23, CL19, ANPKT24, GLM+24b, CL21].

In Table 1, we compare our mercurial signature to related constructions. The closest work to ours is Mir et al. [MBG+23], and we highlight the differences. In particular, our construction supports thresholdization and DAC where [MBG+23] did not. Supporting DAC increases the size of public keys from [MBG+23].

Schemes	[CL19]	[GLM+24b]	[ANPKT24]	[MBG+23]	Ours (Figure 4)
$\lvert\mathsf{pp}\rvert$	$O(1^\lambda)$	$4\ell(\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_2\rvert)$	$O(1^\lambda)$	$O(1^\lambda)$	$O(1^\lambda)$
$\lvert\mathsf{pk}\rvert$	$\ell \cdot (\lvert\mathbb{G}_2\rvert)$	$2\ell \cdot (\lvert\mathbb{G}_2\rvert)$	$\ell \cdot (\lvert\mathbb{G}_2\rvert)$	$\ell \cdot (\lvert\mathbb{G}_2\rvert)$	$\ell \cdot (\lvert\mathbb{G}_2\rvert + 2\lvert\mathbb{G}_1\rvert)^{\dagger\dagger}$
$\lvert\sigma\rvert$	$3 \cdot \lvert\mathbb{G}_1\rvert$	$3 \cdot \lvert\mathbb{G}_1\rvert$	$3 \cdot \lvert\mathbb{G}_1\rvert$	$3 \cdot \lvert\mathbb{G}_1\rvert$	$3 \cdot \lvert\mathbb{G}_1\rvert$
$O(\mathsf{Verify})$	$(\ell + 3) \cdot e$	$(5\ell + 3) \cdot e$	$(\ell + 3) \cdot e$	$(4\ell + 3) \cdot e$	$(4\ell + 3) \cdot e$
Threshold	×	×	✓	×†	✓
Non-interactive	N/A‡	N/A‡	×	✓	✓
Supports DAC	✓	✓	✓	×	✓

Table 1: Concrete parameter comparison.

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 $\lvert\mathsf{pp}\rvert$ (beyond the security parameter) implies trusted setup (any non-trusted setup can be sampled in the ROM). $O(\mathsf{Verify})$ is the complexity of signature verification and $X \cdot e$ indicates that X pairings must 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 [MBG+23].

1.2 Technical Overview

Our threshold mercurial signature takes the approach of Crites et al. [CKP+23] to thresholdize the secret key using Shamir secret sharing [Sha79]. 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 [DS19].

Our scheme is based on the key-homomorphic signature construction in [MBG+23][1]. In this construction, a message[2] $(\vec{M}, \vec{N})$ with $\vec{M} = (M_1, M_2)$ and a tag[3] $\vec{T} = (T_1, T_2)$ 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 [MBG+23] 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 [MBG+23], 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 [MBG+23]. 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[4]. 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-

[1] [MBG+23] uses bilinear source groups $\mathbb{G}_1$ and $\mathbb{G}_2$ generated by P and $\hat{P}$ respectively

[2] 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.

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

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

DIW) [BBH+24], ISO's group signatures [ISO13], TCG's Direct Anonymous Attestation protocols [TCG19], and W3C's Decentralized Identifiers [W3C22]. Several implementations and specifications have emerged, such as Hyperledger AnonCreds [HYP23], Microsoft's uProve [PZ13], and IBM's idemix [CH02]. Delegatable anonymous credentials (DACs) were first realized by Chase and Lysyanskaya [CL06] but were not made efficient for long chains until the work of Belenkiy et al. [BCC+09], which relied on Groth-Sahai proofs [GS08]. Since Groth-Sahai proofs are often considered inefficient, Crites and Lysyanskaya [CL19] 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 [Nak09, MXC+16, YMR+19, GGN16, PCCY22]. 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 [FLL25], where the key used to show a signature is shared across multiple devices.

Mercurial Signatures. Mercurial signatures were introduced in [CL19] as a combination of equivalence classes on messages [FHS19] and equivalence classes on public keys [BHKS18]. Mercurial signatures belong to a broader family of schemes with randomizable public keys, including *signatures with re-randomizable keys* [FKM+16], *key-blinded signatures*, [ESS21] *signatures with honestly randomized keys*, [DFL19] and *updatable signatures* [CRS+21]. For a disambiguation, see the work of Celi et al. [CGH+24]. Mercurial signatures were initially used to realize more efficient DACs that out-performed the previous construction built using Groth-Sahai proofs [BCC+09], but have also been used in other applications, such as contact tracing [GL24]. Some mercurial schemes [MBG+23, CL19] suffer from a limitation: signature unlinkability only holds when the signer does not collude with the adversary. Griffy et al. [GLM+24b] strengthened the definition and construction of mercurial signatures from [CL19] to support privacy against malicious signers, though this construction requires a structured common reference string (SRS).

Many mercurial schemes are also *structure-preserving signatures* [AFG+10]. 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 [GS08]. 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 [GKR08]). Moreover,

Groth-Sahai proofs are *randomizable*, which is what allowed the efficient construction of DACs in [BCC+09].

Threshold Mercurial Signatures. Threshold mercurial signatures were first introduced by Abe et al. in [ANPKT24], which adapted [CL19] to the threshold setting to create stronger anonymity and unforgeability properties for delegatable anonymous credentials. While their construction does not require an SRS (unlike [GLM+24b]), 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. [MBG+23] which introduced *aggregatable* mercurial signatures. However, their scheme did not support the signing of public keys as messages [GL24], which limited their application to issuer-hiding [BEK+21] and multi-authority [HP22] 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 [**?**].

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 [Sha79]. 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 [MBG+23, CL19, CKP+23], 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 [GS08] and allowed for efficient DAC constructions [BCC+09]. However, it was later shown that using only a single source group leads to impossibility results concerning signatures size [AGHO11]. This limitation was mitigated by defining message spaces over multiple source groups [Gha16] (*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* [CKP+23].

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 [MBG+23] (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 [MBG+23]. 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 [CKP+23][5]. 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 ($\mathcal{T}^{\vec{N}}$) [MBG+23]). *$\mathcal{T}^{\vec{N}} \subset (\mathbb{G}_1)^\ell$ is a tag space if for all $\vec{T} = (T_1, \ldots, T_\ell) \in \mathcal{T}^{\vec{N}}$, $\exists\ \gamma \in \mathbb{Z}_p^\times$, $\vec{\rho} = (\rho_1, \ldots, \rho_\ell)$ such that $\vec{N} = (N_1, \ldots, N_\ell) \in (\mathbb{G}_2)^\ell$, $h = H(P^{\rho_1}||\ldots||P^{\rho_\ell}||N_1||\ldots||N_\ell)$ and $\vec{T} = (h^{\gamma\rho_1}, \ldots, h^{\gamma\rho_\ell})$.*

Definition 2 (Tag-based DH Message Space ($\mathcal{M}$) [MBG+23]). *$\mathcal{M} \subset (\mathbb{G}_1)^\ell \times (\mathbb{G}_2)^\ell$ is a tag-based Diffie-Hellman (DH) message space if for all $(\vec{M}, \vec{N}) = (M_1, \ldots, M_\ell, N_1, \ldots, N_\ell) \in \mathcal{M}$, $\exists \vec{T} \in \mathcal{T}^{\vec{N}}$ 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 [CKP+23, MBG+23], since h depends on $\vec{T}$ and $\vec{M}$ depends on $(\vec{T}, h)$, to generate messages, we first sample $m_1, \ldots, m_\ell, \rho_1, \ldots, \rho_\ell$ from $\mathbb{Z}_p^\times$. Then, $\vec{N} = (\hat{P}^{m_1}, \ldots, \hat{P}^{m_\ell})$ and $h = H(P^{\rho_1}||\ldots||P^{\rho_\ell}||\hat{N}_1||\ldots||\hat{N}_\ell)$ are computed. $\vec{M}$ can be computed easily from there. This generation is described in the scheme by in the function GenAuxTag in Definition 3 with corresponding verification function, VerifyAux[6]. 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 [ANPKT24], our TMS scheme does not require any interaction between the signers during

[5] This was mentioned w.r.t. indexed message spaces, but the dependency was present in tag-based message spaces in [MBG+23]

[6] The "aux" part of VerifyAux is an artifact from the aggregation property of [MBG+23] which we inherit for consistency. $\mathsf{aux} = \bot$ for our construction.

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.

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 ConvertTag, ConvertSK, ConvertPK, ConvertSig, and ChangeRep functions, the converter is sampled uniformly from $\mathbb{Z}_p^\times$ if omitted. To facilitate the verification of tags, the functions GenAuxTag, VerifyTag, and VerifyAux are included. We described how GenAuxTag and VerifyAux are used to verify tags in Section 2. 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*,* GenAuxTag*,* VerifyAux*,* VerifyTag*,* ParSign*,* ParVerify*,* Reconst*,* Verify*,* ConvertTag*,* ConvertParSK*,* ConvertParPK*,* ConvertPK*,* ConvertSig*,* 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* $1 \leq t \leq n$*, 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.*

– $\mathsf{Reconst}(\vec{\mathsf{pk}}, \vec{T}, (\vec{M}, \vec{N}), \{i, \sigma_i\}_{i \in \mathfrak{T}})) \rightarrow \sigma$*: 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*.*

– $\mathsf{Verify}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma) \rightarrow \{0, 1\}$*: 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.*

– $\mathsf{ConvertTag}(\vec{T}; \gamma) \rightarrow \vec{T}'$*: The tag conversion algorithm takes as input a tag* $\vec{T}$ *and converter* $\gamma \in \mathbb{Z}_p^\times$*, and outputs a new tag* $\vec{T}' \in [\vec{T}]_{\mathcal{R}_\mathcal{T}}$*.*

– $\mathsf{ConvertParSK}(\mathsf{sk}_i; \omega) \rightarrow \mathsf{sk}_i'$*: 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* $\mathsf{sk}_i' \in [\mathsf{sk}_i]_{\mathcal{R}_{\mathcal{SK}}}$*.*

– $\mathsf{ConvertParPK}(\mathsf{pk}_i; \omega) \rightarrow \mathsf{pk}_i'$*: 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* $\mathsf{pk}_i' \in [\mathsf{pk}_i]_{\mathcal{R}_\mathcal{K}}$*.*

– $\mathsf{ConvertPK}(\mathsf{pk}_0; \omega) \rightarrow \mathsf{pk}_0'$*: 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* $\mathsf{pk}_0' \in [\mathsf{pk}_0]_{\mathcal{R}_\mathcal{K}}$*.*

– $\mathsf{ConvertSig}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma; \omega) \rightarrow \sigma'$*: 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* $\mathsf{pk}_0' = \mathsf{ConvertPK}(\mathsf{pk}_0; \omega)$

– $\mathsf{ChangeRep}(\mathsf{pk}_0, \vec{T}, (\vec{M}, \vec{N}), \sigma; (\mu, \nu)) \rightarrow (\vec{T}', (\vec{M}', \vec{N}'), \sigma')$*: 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* $\mathsf{TMS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{GenAuxTag}, \mathsf{VerifyAux}, \mathsf{VerifyTag}, \mathsf{ParSign}, \mathsf{ParVerify}, \mathsf{Reconst}, \mathsf{Verify}, \mathsf{ConvertTag}, \mathsf{ConvertParSK}, \mathsf{ConvertParPK}, \mathsf{ConvertPK}, \mathsf{ConvertSig}, \mathsf{ChangeRep})$ *is correct if, for all* $\lambda, n, t \in \mathbb{N}$ *such that* $1 \leq t \leq n$*, 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.* $1 \le t \le n$, *and for all* PPT *adversaries* $\mathcal{A}$,

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

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

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

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

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

1 : **if** $i \notin [n]$ **then**
2 : **return** $\perp$
3 : **else**
4 : $\sigma_i \leftarrow \mathsf{ParSign}(\mathsf{sk}_i, \vec{\rho}, \mathsf{aux}, (\vec{M}, \vec{N}))$
5 : $Q \leftarrow Q \cup (\vec{M}, \vec{N})$
6 : **return** σ_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* $1 \le t \le n$, *and for all* PPT *adversaries* $\mathcal{A}$,

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

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,

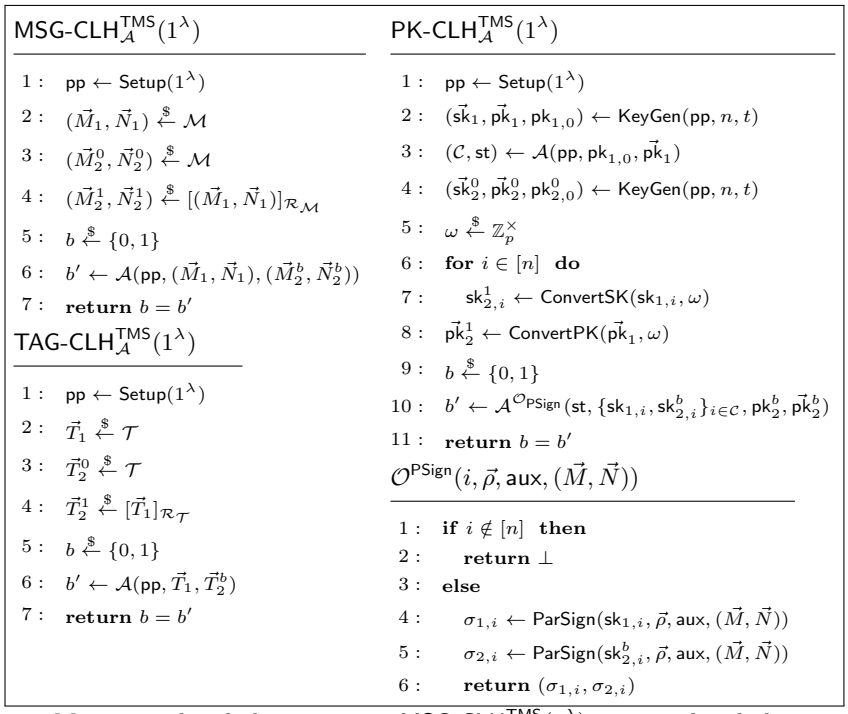

Fig. 2: Message class-hiding game MSG-CLH$^{\mathsf{TMS}}_{\mathcal{A}}(1^\lambda)$, tag class-hiding game TAG-CLH$^{\mathsf{TMS}}_{\mathcal{A}}(1^\lambda)$, public key class-hiding game PK-CLH$^{\mathsf{TMS}}_{\mathcal{A}}(1^\lambda)$ for TMS scheme.

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 TAG-CLH$^{\mathsf{TMS}}_{\mathcal{A}}(1^\lambda)$ *is defined in Figure 2.*

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

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

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

Definition 9 (Origin-Hiding). *A threshold mercurial signature scheme* (Setup, KeyGen, GenAuxTag, VerifyAux, VerifyTag, ParSign, ParVerify, Reconst, Verify, ConvertTag, ConvertParSK, ConvertParPK, ConvertPK, ConvertSig,

$\mathsf{ChangeRep}$) *is origin-hiding if, for all* $\lambda, n, t \in \mathbb{N}$ *such that* $1 \leq t \leq n$*, 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 [MBG+23], do not support the signing of public keys. This was an informal property of [CL19]

that was later formalized as *cross-scheme correctness* in [GL24]. We define a similar property 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 [MBG+23])

Before explaining our threshold construction, we present the mercurial signature construction from [MBG+23] that our scheme is based on. Although the original scheme in [MBG+23] 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 [MBG+23]. 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 [MBG+23].

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

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.*

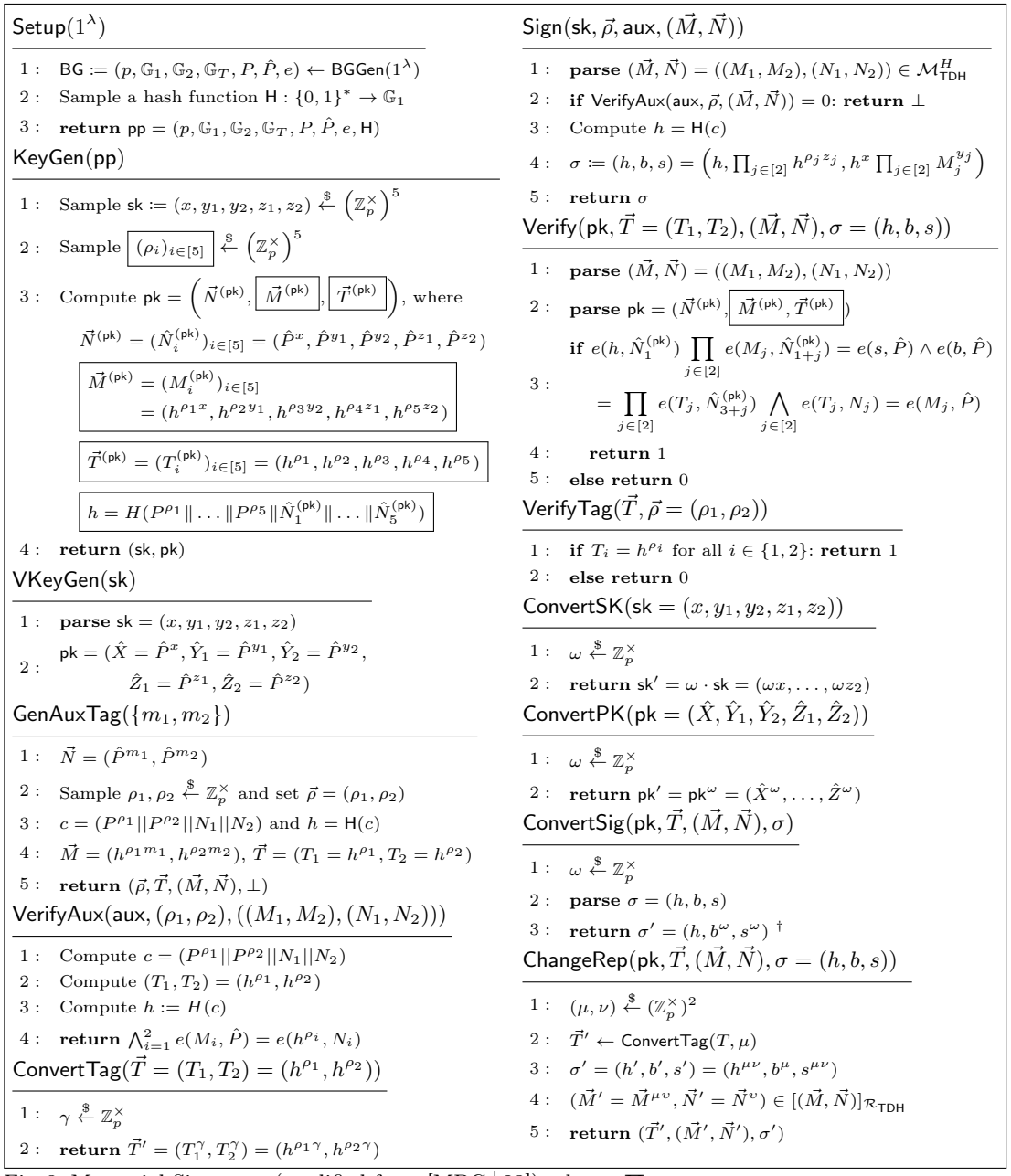

Fig. 3: Mercurial Signature (modified from [MBG+23]) scheme $\mathbf{\Pi}_{\mathsf{MBGLS}}$.

† 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 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* (EUF-CtMA) *as per Definition 24 (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 22 (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 [MBG+23]. For the cross-scheme correct version of $\mathbf{\Pi}_{\mathsf{MBGLS}}$, Theorems 1 and 3 to 5 follow from [MBG+23]. 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 2 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 7 and the modified mercurial signature scheme $\mathbf{\Pi}_{\mathsf{MBGLS}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{VKeyGen}, \mathsf{GenAuxTag}, \mathsf{VerifyAux}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{VerifyTag}, \mathsf{ConvertTag}, \mathsf{ChangeRep}, \mathsf{ConvertSK}, \mathsf{ConvertPK}, \mathsf{ConvertSig})$ from Figure 3.

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

TMS for arbitrary ℓ. Construction 2 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,

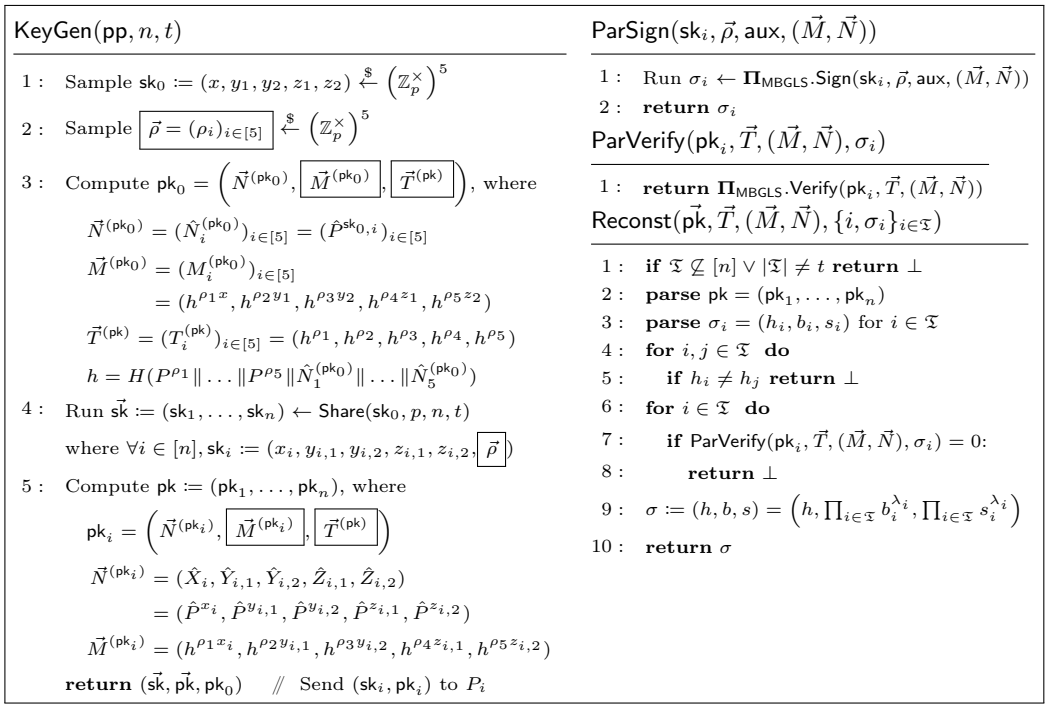

Fig. 4: Construction of threshold mercurial signature scheme $\mathbf{\Pi}_{\mathsf{TMS}}$.

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 (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 7, 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 7, 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 24 (informally described in Definition 10), and assuming that* SSS *is a secure Shamir secret sharing scheme as in Figure 7, 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 22 (informally described in Definition 11), and that* SSS *is a secure Shamir secret sharing scheme as in Figure 7, 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 [MBG+23], 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 [MBG+23] 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 [GLM+24b].

We outline the high-level functionality of our TDAC scheme in Definition 13. 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 $\mathsf{IssKeyGen}$[7] 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.

[7] 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.

4.1 Syntax and Security Definition

Definition 13 (Threshold Delegatable Anonymous Credentials). *A threshold delegatable anonymous credentials scheme* TDAC *consists of the following* PPT *algorithms (*Setup*,* IssKeyGen*,* UserKeyGen*,* Issue*,* Receive*,* CredComb*,* UIssue*,* UReceive*,* UCredComb*,* Prove*,* Verify*), which have the following syntax:*

- $\mathsf{Setup}(1^\lambda, 1^L, 1^n, 1^t) \to \mathsf{pp}$*: 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* Receive *takes as input receiver a partial secret key*[8] 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* 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* 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 threshold delegatable anonymous credentials scheme TDAC *must satisfy the properties of: correctness (Definition 14), anonymity (Definition 15), and unforgeability (Definition 16).*

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

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 5.

- $\mathcal{O}^{\mathsf{CreateHI}}$: 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.
- $\mathcal{O}^{\mathsf{IssueToAdv}}$: In the anonymity game, we allow the adversary to receive credentials as an issuer using the $\mathcal{O}^{\mathsf{IssueToAdv}}$ oracle.
- $\mathcal{O}^{\mathsf{IssueFromAdv}}$: We allow the adversary to issue credentials to honest issuers using the $\mathcal{O}^{\mathsf{IssueFromAdv}}$ oracle[9]. In this oracle, the adversary specifies an issuer set (id_R) to receive a credential at a specified level (L').
- $\mathcal{O}^{\mathsf{IssueFromAdv}}$: In the $\mathcal{O}^{\mathsf{IssueFromAdv}}$ oracle, the adversary receives a credential from an honest issuer set (id_I).
- $\mathcal{O}^{\mathsf{ProveToAdv}}$: 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.
- $\mathcal{O}^{\mathsf{UIssueFromAdv}}$ 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 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

[9] Diverging from the definition in [GLM+24b], 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.

papers since their introduction in 2019 [CL19]. 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.

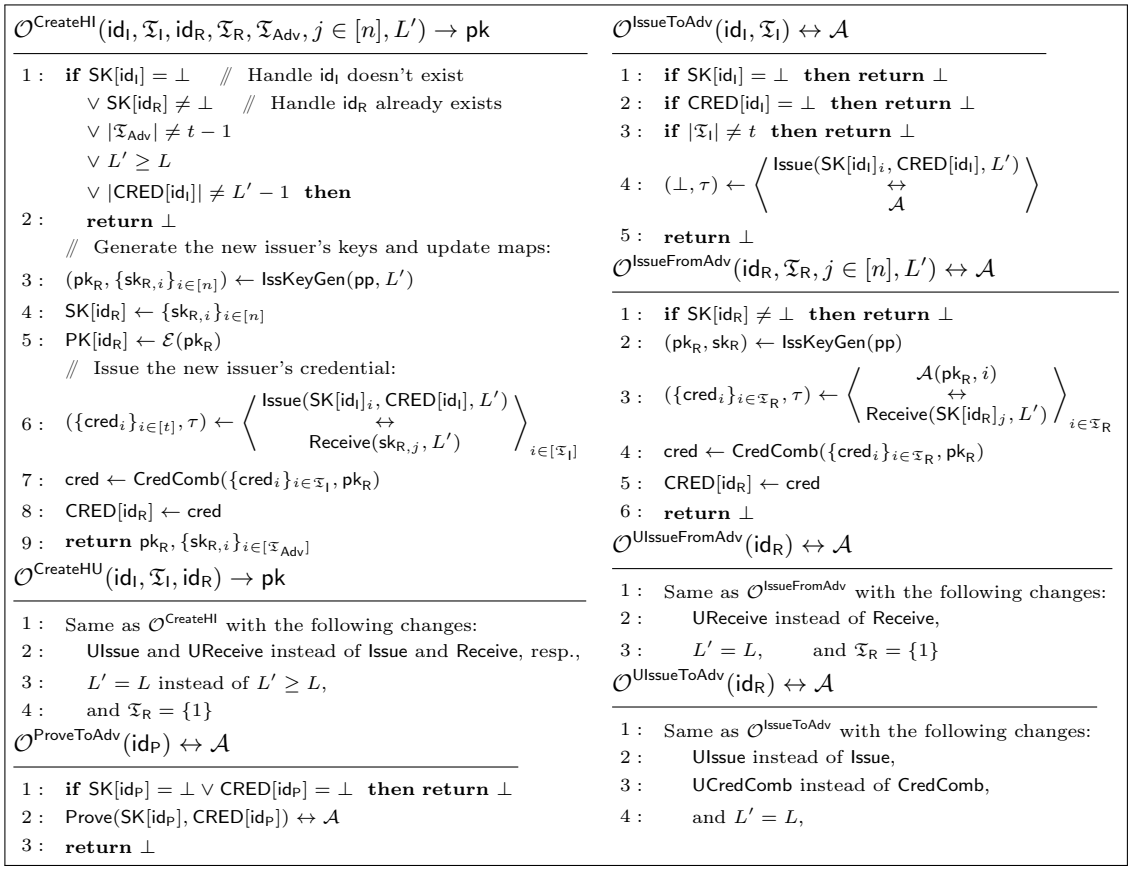

Fig. 5: Definition of Oracles for TDAC scheme.

Correctness. We define correctness for our threshold DAC scheme (in Definition 14).

Definition 14 (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* $\mathsf{cred}_0 = \bot$, *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 creden-*

tials, $\{\mathsf{cred}_{L,j} \in (\mathsf{UIssue}(\mathsf{sk}_{L-1,j}, \mathsf{cred}_{i-1}, i-1) \leftrightarrow \mathsf{UReceive}(\mathsf{pk}_L, i))\}_{j\in\mathfrak{T}}$*, 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 15. 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. 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 15 (Anonymity). *A threshold delegatable anonymous credentials scheme* $\mathsf{TDAC} = (\mathsf{Setup}, \mathsf{IssKeyGen}, \mathsf{UserKeyGen}, \mathsf{Issue}, \mathsf{Receive}, \mathsf{CredComb}, \mathsf{UIssue}, \mathsf{UReceive}, \mathsf{UCredComb}, \mathsf{Prove}, \mathsf{Verify})$ *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* $\mathsf{ANON}^{\mathsf{TDAC}}_{\mathcal{A}}(1^\lambda)$ *(defined by* $\mathsf{Adv}^{\mathsf{ANON}}_{\mathsf{TDAC},\mathcal{A}}$*) is negligible.*

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

where the anonymity game $\mathsf{ANON}^{\mathsf{TDAC}}_{\mathcal{A}}(1^\lambda)$ *is defined in Figure 6 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 5.*

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

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** $b = b'$

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

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. 6: Anonymity game $\mathsf{ANON}^{\mathsf{TDAC}}_{\mathcal{A}}(1^\lambda)$ and Unforgeability game $\mathsf{unForge}^{\mathsf{TDAC}}_{\mathcal{A}}(1^\lambda)$ for TDAC scheme. The oracles $\mathcal{O}$ is described in Figure 5.

Unforgeability. We define unforgeability for our threshold DAC scheme in Definition 16. 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 16 (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* $\mathsf{Adv}^{\mathsf{unForge}}_{\mathsf{TDAC},\mathcal{A}}$ *in* $\mathsf{unForge}^{\mathsf{TDAC}}_{\mathcal{A}}(1^\lambda)$ *is negligible.* $\mathcal{A}$ *is given the oracles* $\mathcal{O}^{\mathsf{CreateHI}}$*,* $\mathcal{O}^{\mathsf{CreateHU}}$*, and* $\mathcal{O}^{\mathsf{ProveToAdv}}$*, from Figure 5 labeled as* $\mathcal{O}_{\mathsf{Unf}}$*.*

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

where the unforgeability game $\mathsf{unForge}^{\mathsf{TDAC}}_{\mathcal{A}}(1^\lambda)$ *is defined in Figure 6.*

4.2 Construction of TDAC

In Construction 1, 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 7.

Construction 1 (Threshold Delegatable Anonymous Credentials) *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. *Run* $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{TMS.KeyGen}(\mathsf{pp}_{L'}, n, t)$[10]
  2. *Output* $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]})$
- $\mathsf{UserKeyGen}(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{sk})$

[10] 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 $\vec{\rho}$.

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{TMS.KeyGen}(\mathsf{pp}_L, 1, 1)$
2. *Output* $(\mathsf{pk}, \mathsf{sk})$

– $\langle \mathsf{Issue}(\mathsf{sk}_{\mathsf{I},i}, \mathsf{cred}_{\mathsf{I}}, L') \leftrightarrow \mathsf{Receive}(\mathsf{sk}_{\mathsf{R}}, L') \rangle \rightarrow \mathsf{cred}_{\mathsf{R},i}$*:*
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 [MBG+23][11]*
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.*

– $\mathsf{CredComb}(\{\mathsf{cred}_i\}_{i \in \mathfrak{T}}, \mathsf{pk}_{\mathsf{R}}) \rightarrow \mathsf{cred}_{\mathsf{R}}$*:*
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*.*

– $\langle \mathsf{UIssue}(\mathsf{sk}_{\mathsf{I},i}, \mathsf{cred}_{\mathsf{I}}) \leftrightarrow \mathsf{UReceive}(\mathsf{sk}_{\mathsf{R}}) \rangle \rightarrow \mathsf{cred}_{\mathsf{R}}$*: 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).*

– $\mathsf{UCredComb}(\{\mathsf{cred}_{\mathsf{R},i}\}_{i \in \mathfrak{T}}, \mathsf{pk}_{\mathsf{R}}) \rightarrow \mathsf{cred}_{\mathsf{R}}$*: This is exactly same as* $\mathsf{CredComb}(\{\mathsf{cred}_{\mathsf{R},i}\}_{i \in \mathfrak{T}}, \mathsf{pk}_{\mathsf{R}}) \rightarrow \mathsf{cred}_{\mathsf{R}}$*.*

– $\langle \mathsf{Prove}(\mathsf{sk}_{\mathsf{P}}, \mathsf{cred}_{\mathsf{P}}, L') \leftrightarrow \mathsf{Verify}(\mathsf{pk}, L') \rangle \rightarrow b$*:*
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* $\mathsf{cred}_{\mathsf{P}}$ *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,* $\mathsf{cred}' = (\mathsf{cred}'_1 || \ldots || \mathsf{cred}'_L)$ *where* $\mathsf{cred}'_i = (\mathsf{pk}'_i, \sigma'_i)$*, and performs an interactive proof of knowledge (such as a Camenisch-Stadler proof [CS97]) 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*.*

4.3 Security Analysis of TDAC

We formally state the security of our threshold DAC scheme in Construction 1 in Theorems 15 to 17, which are proven in Appendix C.

[11] 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 $\vec{T} \in \mathcal{T}^{\vec{T}}$.

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 Construction 1 is a correct threshold delegatable anonymous credentials scheme as per Definition 14.*

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 Construction 1 is an unforgeable threshold delegatable anonymous credentials scheme* TDAC *as per Definition 16.*

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 Construction 1 is an anonymous threshold delegatable anonymous credentials scheme as per Definition 15.*

References

AFG+10. Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 209–236. Springer, Berlin, Heidelberg, August 2010.

AGHO11. Masayuki Abe, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Optimal structure-preserving signatures in asymmetric bilinear groups. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 649–666. Springer, Berlin, Heidelberg, August 2011.

ANO+25. Masayuki Abe, Masaya Nanri, Miyako Ohkubo, Octavio Perez Kempner, Daniel Slamanig, and Mehdi Tibouchi. Scalable mixnets from two-party mercurial signatures on randomizable ciphertexts. In *Computer Security – ESORICS 2025*, 2025.

ANPKT24. Masayuki Abe, Masaya Nanri, Octavio Perez-Kempner, and Mehdi Tibouchi. Interactive threshold mercurial signatures and applications. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part III*, volume 15486 of *LNCS*, pages 69–103. Springer, Singapore, December 2024.

BBH+24. Carsten Baum, Olivier Blazy, Jaap-Henk Hoepman, Anja Lehmann, Anna Lysyanskaya, René Mayrhofer, Hart Montgomery, Ngoc Khanh Nguyen, abhi shelat, Daniel Slamanig, Søren Eller Thomsen, Jan Camenisch, Eysa Lee, Bart Preneel, Stefano Tessaro, and Carmela Troncoso. 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`.

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

BEK+21. Jan Bobolz, Fabian Eidens, Stephan Krenn, Sebastian Ramacher, and Kai Samelin. Issuer-hiding attribute-based credentials. In Mauro Conti, Marc Stevens, and Stephan Krenn, editors, *CANS 21*, volume 13099 of *LNCS*, pages 158–178. Springer, Cham, December 2021.

BFR24. Balthazar Bauer, Georg Fuchsbauer, and Fabian Regen. On proving equivalence class signatures secure from non-interactive assumptions. In Qiang Tang and Vanessa Teague, editors, *PKC 2024, Part I*, volume 14601 of *LNCS*, pages 3–36. Springer, Cham, April 2024.

BHKS18. Michael Backes, Lucjan Hanzlik, Kamil Kluczniak, and Jonas Schneider. Signatures with flexible public key: Introducing equivalence classes for public keys. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 405–434. Springer, Cham, December 2018.

CDHK15. Jan Camenisch, Maria Dubovitskaya, Kristiyan Haralambiev, and Markulf Kohlweiss. Composable and modular anonymous credentials: Definitions and practical constructions. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 262–288. Springer, Berlin, Heidelberg, November / December 2015.

CGH+24. Sofía Celi, Scott Griffy, Lucjan Hanzlik, Octavio Perez-Kempner, and Daniel Slamanig. SoK: Signatures with randomizable keys. In Jeremy Clark and Elaine Shi, editors, *FC 2024, Part II*, volume 14745 of *LNCS*, pages 160–187. Springer, Cham, March 2024.

CH02. Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In *Conference on Computer and Communications Security*, 2002.

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

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

CKP+23. Elizabeth C. Crites, Markulf Kohlweiss, Bart Preneel, Mahdi Sedaghat, and Daniel Slamanig. Threshold structure-preserving signatures. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part II*, volume 14439 of *LNCS*, pages 348–382. Springer, Singapore, December 2023.

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

CL04. Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer, Berlin, Heidelberg, August 2004.

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

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

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

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

CRS+21. Valerio Cini, Sebastian Ramacher, Daniel Slamanig, Christoph Striecks, and Erkan Tairi. Updatable signatures and message authentication codes.

In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 691–723. Springer, Cham, May 2021.

CS97. Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In Burton S. Kaliski, Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 410–424. Springer, Berlin, Heidelberg, August 1997.

CvH91. David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 257–265. Springer, Berlin, Heidelberg, April 1991.

Des88. Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 120–127. Springer, Berlin, Heidelberg, August 1988.

DF90. Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer, New York, August 1990.

DFL19. Poulami Das, Sebastian Faust, and Julian Loss. A formal treatment of deterministic wallets. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 651–668. ACM Press, November 2019.

DKL+23. Jack Doerner, Yashvanth Kondi, Eysa Lee, Abhi Shelat, and LaKyah Tyner. Threshold bbs+ signatures for distributed anonymous credential issuance. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 773–789, 2023.

DS19. David Derler and Daniel Slamanig. Key-homomorphic signatures: definitions and applications to multiparty signatures and non-interactive zero-knowledge. *DCC*, 87(6):1373–1413, 2019.

eID24. 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.

ESS21. Edward Eaton, Douglas Stebila, and Roy Stracovsky. Post-quantum key-blinding for authentication in anonymity networks. In Patrick Longa and Carla Ràfols, editors, *LATINCRYPT 2021*, volume 12912 of *LNCS*, pages 67–87. Springer, Cham, October 2021.

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

FKM+16. Nils Fleischhacker, Johannes Krupp, Giulio Malavolta, Jonas Schneider, Dominique Schröder, and Mark Simkin. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part I*, volume 9614 of *LNCS*, pages 301–330. Springer, Berlin, Heidelberg, March 2016.

FLL25. Andrea Flamini, Eysa Lee, and Anna Lysyanskaya. Multi-holder anonymous credentials from BBS signatures. 2025.

GGN16. Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 2016*, volume 9696 of *LNCS*, pages 156–174. Springer, Cham, June 2016.

Gha16. Essam Ghadafi. Short structure-preserving signatures. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 305–321. Springer, Cham, February / March 2016.

GKR08. Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 113–122. ACM Press, May 2008.

GL24. Scott Griffy and Anna Lysyanskaya. PACIFIC: Privacy-preserving automated contact tracing featuring integrity against cloning. *CiC*, 1(2):12, 2024.

GLM+24a. Scott Griffy, Anna Lysyanskaya, Omid Mir, Octavio Perez Kempner, and Daniel Slamanig. Delegatable anonymous credentials from mercurial signatures with stronger privacy. Cryptology ePrint Archive, Report 2024/1216, 2024.

GLM+24b. Scott Griffy, Anna Lysyanskaya, Omid Mir, Octavio Perez-Kempner, and Daniel Slamanig. Delegatable anonymous credentials from mercurial signatures with stronger privacy. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part II*, volume 15485 of *LNCS*, pages 296–325. Springer, Singapore, December 2024.

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

HP22. Chloé Hébant and David Pointcheval. Traceable constant-size multi-authority credentials. In Clemente Galdi and Stanislaw Jarecki, editors, *SCN 22*, volume 13409 of *LNCS*, pages 411–434. Springer, Cham, September 2022.

HYP23. Hyperledger anoncreds, 2023. `https://hyperledger.github.io/anoncreds-spec/`.

ISO13. Iso: Information technology - security techniques - anonymous digital signatures - part 2: Mechanisms using a group public key. Technical report, International Organization for Standardization, Geneva, Switzerland, 2013. `https://www.iso.org/standard/56916.html`.

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

MSM24. Omid Mir, Daniel Slamanig, and René Mayrhofer. Threshold delegatable anonymous credentials with controlled and fine-grained delegation. *IEEE Transactions on Dependable and Secure Computing*, 21(4):2312–2326, 2024.

MXC+16. Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 31–42. ACM Press, October 2016.

Nak09. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

PCCY22. Shimin Pan, Kwan Yin Chan, Handong Cui, and Tsz Hon Yuen. Multi-signatures for ECDSA and its applications in blockchain. In Khoa Nguyen, Guomin Yang, Fuchun Guo, and Willy Susilo, editors, *ACISP 22*, volume 13494 of *LNCS*, pages 265–285. Springer, Cham, November 2022.

PM23. Colin Putman and Keith M. Martin. Selective delegation of attributes in mercurial signature credentials. In Elizabeth A. Quaglia, editor, *19th IMA*

International Conference on Cryptography and Coding, volume 14421 of *LNCS*, pages 181–196. Springer, Cham, December 2023.

PZ13. Christian Paquin and Greg Zaverucha. U-prove cryptographic specification v1.1. Technical report, Microsoft Corporation, 2013.

SAB+19. Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, and George Danezis. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. In *NDSS 2019*. The Internet Society, February 2019.

Sha79. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

Sho97. Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Berlin, Heidelberg, May 1997.

TCG19. Trusted platform module library part 1: Architecture. Technical report, Trusted Computing Group, 2019.

W3C22. Decentralized identifiers (dids) v1.0. Technical report, World Wide Web Consortium, 2022.

YMR+19. Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. HotStuff: BFT consensus with linearity and responsiveness. In Peter Robinson and Faith Ellen, editors, *38th ACM PODC*, pages 347–356. ACM, July / August 2019.