# Double Issuer-Hiding Attribute-Based Credentials From Tag-Based Aggregatable Mercurial Signatures

Rui Shi, Yang Yang, Yingjiu Li, Huamin Feng, Guozhen Shi, Hwee Hwa Pang, and Robert H. Deng, *Fellow, IEEE*

***Abstract*—Attribute-based anonymous credentials offer users fine-grained access control in a privacy-preserving manner. However, in such schemes obtaining a user's credentials requires knowledge of the issuer's public key, which obviously reveals the issuer's identity that must be hidden from users in certain scenarios. Moreover, verifying a user's credentials also requires the knowledge of issuer's public key, which may infer the user's private information from their choice of issuer. In this article, we introduce the notion of double issuer-hiding attribute-based credentials (DIHAC) to tackle these two problems. In our model, a central authority can issue public-key credentials for a group of issuers, and users can obtain attribute-based credentials from one of the issuers without knowing which one it is. Then, a user can prove that their credential was issued by one of the authenticated issuers without revealing which one to a verifier. We provide a generic construction, as well as a concrete instantiation for DIHAC based on structure-preserving signatures on equivalence classes (JOC's 19) and a novel primitive which we call *tag-based aggregatable mercurial signatures*. Our construction is efficient without relying on zero-knowledge proofs. We provide rigorous evaluations on personal laptop and smartphone platforms, respectively, to demonstrate its practicability.**

***Index Terms*—Anonymous credentials, mercurial signatures, privacy-preserving, issuer-hiding.**

Manuscript received 16 August 2022; revised 7 July 2023; accepted 5 September 2023. Date of publication 11 September 2023; date of current version 11 July 2024. The work of Yang Yang and Robert Deng was supported in part by the National Natural Science Foundation of China under Grant 62372110, in part by the Fujian Provincial Natural Science of Foundation under Grant 2023J02008, and in part by Lee Kong Chian Chair Professor Fund and AXA Research Fund. The work of Yingjiu Li was supported by Ripple University Blockchain Research Initiative. The work of Huamin Feng was supported by the National Defense Basic Research Program of China under Grant JCKY2019102C001. The work of Hwee Hwa Pang was supported by Lee Kong Chian Chair Professor Fund. *(Corresponding author: Yang Yang.)*

Rui Shi is with the School of Cyber Science and Technology, Shandong University, Qingdao, Shandong 266237, China, and also with Beijing Electronic Science and Technology Institute, Beijing 100070, China (e-mail: ruishi_mail@126.com).

Yang Yang is with the School of Computing and Information Systems, Singapore Management University, Singapore 188065 (e-mail: yang.yang.research@gmail.com).

Yingjiu Li is with the Department of Computer and Information Science, University of Oregon, Eugene, OR 97403 USA (e-mail: yingjiul@uoregon.edu).

Huamin Feng and Guozhen Shi are with the Institute of Information Security, Beijing Electronic Science and Technology Institute, Beijing 100070, China (e-mail: fenghm@besti.edu.cn; sgz1974@163.com).

Hwee Hwa Pang and Robert H. Deng are with the School of Computing and Information Systems, Singapore Management University, Singapore 188065 (e-mail: hhpang@smu.edu.sg; robertdeng@smu.edu.sg).

This article has supplementary downloadable material available at https://doi.org/10.1109/TDSC.2023.3314019, provided by the authors.

Digital Object Identifier 10.1109/TDSC.2023.3314019

## I. Introduction

ATTRIBUTE-BASED anonymous credentials (ABCs) is a primary cryptography tool for the secure use of digital services, preserving the privacy of users while providing fine-grained authentication services. A typical ABCs scheme involves three types of entities: issuer, user, and verifier. A user obtains credentials from an issuer on a set of attributes that describe the user's personal information. The user then presents the credentials to a verifier anonymously by disclosing a selected subset of the user's attributes. Multiple presentations of the same user can only be linked through the disclosed attributes. ABCs protect user privacy and support fine-grained access control due to selective disclosure of users' attributes for credential verification. After ABCs were introduced [1], a series of ABCs schemes were proposed to achieve various functionalities and optimizations, including selective disclosure credentials [32], [34], [35], [36], [37], [38], [39], [41], [42], [45], [46], [47], delegatable credentials [23], [24], [25], [26], [27], [29], [31], [33], keyed-verification credentials [3], [4], [7], updatable credentials [5], [6], and decentralized credentials [2], [37]. Now, ABCs have been widely applied to various anonymous authentication systems, such as electronic tickets systems [8], single sign-on systems [9], permissioned blockchain systems [10], [29], direct anonymous attestation [11], enhanced privacy ID [12], and self-sovereign identity systems [13], which require minimal collection of users' personal information due to privacy regulations, such as General Data Protection Regulations (GDPR) [14] and California Consumer Privacy Act (CCPA) [15].

*Issuer-Hiding From Users:* All the anonymous credential schemes aforementioned have a common feature: users know the issuer's public key when obtaining their credentials. While this seems to be a natural premise, there are scenarios where the issuer's identity must be hidden from users to protect the issuer's privacy. For example, consider deploying anonymous credentials in a blockchain system to enable the automatic issuance of users' credentials through smart contracts, where credential issuers are integrated into the blockchain nodes. Due to the complexity of distributed networks, there is no guarantee that each node (issuer) can always provide services online in real time. A solution to solve this issue is to deploy multiple nodes (issuers) with the same functionality to provide the issuance service, and this should be transparent to users; each user only needs to confirm that their issuer is legitimate without knowing the number and identities of issuers in the system. Another

instance is that multiple issuers should be deployed in a large-scale electronic identity system in order to improve the stability of the system and avoid a single point of failure. In this case, hiding the information of issuers from users and potential adversaries helps reduce the probability of targeted attacks. An fashionable instance is car sharing, where each car owner plays the role of the issuer, and each car sharer plays the role of a user. To transfer access to a car from its owner to a car sharer, the car owner may issue credentials without revealing their identity for privacy protection.

*Issuer-Hiding From Verifiers:* Another concern that has not been commonly addressed in existing research on ABCs is that each user is required to reveal an issuer's public key to a verifier when presenting their credentials, and this enables the verifier to infer the user's private information based on their choice of issuer in certain scenarios. In an electronic identity system, for example, a single issuer is set up for issuing credentials to all users in each administrative area. Fine-grained administrative areas may be arranged to increase the efficiency and manageability of the whole system. However, in such case, a user revealing their issuer's public key to a verifier when presenting their credentials enables the verifier to infer the administrative area of the user. Another instance is that a vehicular ad-hoc networks (VANET) system allows each vehicle's manufacturer to issue credentials to the vehicle. When a vehicle reveals its manufacturer's public key to a verifier for anonymous authentication in VANET, it reveals its brand unnecessarily.

*Drawbacks of Existing Schemes:* The main drawback of existing ABCs schemes is that none was designed to achieve both issuer-hiding from users and issuer-hiding from verifiers. Recently, though, Bobolz et al. [16], Conolly et al. [20], and Bosk et al. [30] independently proposed definitions of issuer-hiding (or signer-hiding) from verifiers and developed concrete anonymous credential schemes based on their definitions. Unfortunately, their schemes do not support issuer-hiding from users. Another trivial solution is to deploy multiple issuers who share the same issuing key, but this approach has two significant drawbacks. First, if any of the multiple issuers is corrupted, the issuing keys of all issuers are compromised. Second, sharing issuer keys is not feasible in many application scenarios. For instance, independent issuers should be deployed in a large-scale electronic identity system based on their administrative divisions. Likewise, in the case of vehicular ad-hoc networks (VANET), each manufacturer should manage an independent credential issuer.

### A. Our Contributions

In this paper, we advance the state-of-the-art by presenting a double issuer-hiding attribute-based credential scheme (DIHAC), which supports issuer-hiding from both users and verifiers while retaining other desired features of anonymous credentials, including anonymity, unforgeability, attribute-based, and selective disclosure. Specifically, our contributions include:

- We introduce a new tag-based aggregatable mercurial signature ($\mathsf{TAM-Sign}$) as a fundamental building block for DIHAC, which is also of independent interest for other applications. Within our $\mathsf{TAM-Sign}$ scheme, multiple attribute signatures $\{\sigma_i\}$ on the same tag $\vec{T}$ can be aggregated into a compact signature $\sigma$. Furthermore, the $\mathsf{TAM-Sign}$ allows a signature $\sigma$ for the attributes on a tag $\vec{T}$ under a public key $pk$ to be transformed into a new unlinkable signature $\sigma'$ for the same attributes on an equivalent but unlinkable tag $\vec{T}'$ under an equivalent but unlinkable public key $pk'$.
- We introduce DIHAC, a new attribute-based anonymous credentials scheme that achieves issuer-hiding from both users and verifiers. We formalize the system model and the security model of DIHAC, including unforgeability, anonymity, and unlinkability. Moreover, we provide a generic construction and an efficient instantiation of DIHAC using our $\mathsf{TAM-Sign}$ scheme in combination with $\mathsf{SPS-EQ}$ scheme [46], and zero-knowledge signature of knowledge (ZKSoK) [23], and prove that the proposed scheme achieves all the security requirements in the formalized security model.
- We implement our DIHAC instantiation at AES-100 b security level, and our source code is available publicly [55]. We evaluate DIHAC on a personal laptop and smartphone, respectively. When the number of user's attributes is set to 10, a user presenting a credential to a verifier takes 41 ms to execute on a laptop and 541 ms to execute on a smartphone. In addition, we highlight its application in an electronic ticket system [8] and a permissioned token system [29] in Supplementary Material A, available online.

## II. Preliminary

### A. Bilinear Pairing

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be cyclic groups of prime order $p$. Let $g$ and $\widetilde{g}$ be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. The mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map if it has three properties: (1) *bilinearity:* $\forall g \in \mathbb{G}_1, \widetilde{g} \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$, we have $e(g^a, \widetilde{g}^b) = e(g, \widetilde{g})^{ab}$. (2) *non-degeneracy:* $e(g, \widetilde{g}) \neq 1_{\mathbb{G}_T}$. (3) *computability:* $e$ can be efficiently computed. We denote a bilinear group $\mathcal{BL} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, p, g, \widetilde{g})$. Our scheme is based on the Type-III pairing [44], which means that there is no efficiently computable homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$.

### B. Pseudorandom Function

A pseudorandom function (PRF) is a deterministic function of a key and an input and consists of the following polynomial-time (PPT) algorithms:

- $\mathsf{PRF.KeyGen}(1^\lambda) \to rk$: On input security parameter $1^\lambda$, outputs a secret key $rk$.
- $\mathsf{PRF.Eval}(rk, str) \to \gamma$: On input a secret key $rk$ and a string $str \in \{0,1\}^*$, outputs a random element $\gamma \in \mathbb{G}$, where $\mathbb{G}$ is a finite set.

A PRF scheme is secure if any PPT adversary cannot distinguish PRF's outputs from a truly random function. In this paper, we use PRF to generate deterministic random scalars.

### C. Class-Hiding

Let $\mathbb{G}_i$ be a prime order group of a bilinear map. $[\vec{M}]_{\mathcal{R}} \subset (\mathbb{G}_i^*)^l$ is a message equivalence class defined based on the equivalence relation: $\mathcal{R}_{\vec{M}} = \{(\vec{M}, \vec{M}') \in (\mathbb{G}^*)^l \times (\mathbb{G}^*)^l | \exists s \in \mathbb{Z}_p^* : \vec{M}' = \vec{M}^s\}$. $(\mathbb{G}_i^*)^l$ is a class-hiding space if for any PPT adversary $\mathcal{A}$ there is a negligible function $\epsilon(\lambda)$ such that:

$$\left| Pr \left[ b^* = b \;\middle|\; \begin{array}{l} b \xleftarrow{R} \{0,1\}; \\ \vec{M} \xleftarrow{R} (\mathbb{G}_i^*)^l; \\ \vec{M}^0 \xleftarrow{R} (\mathbb{G}_i^*)^l; \\ \vec{M}^1 \xleftarrow{R} [\vec{M}]_{\mathcal{R}}; \\ b^* \leftarrow \mathcal{A}(\mathbb{G}_i, \vec{M}, \vec{M}^b). \end{array} \right] - \frac{1}{2} \right| \leqslant \epsilon(\lambda)$$

And, $(\mathbb{G}_i^*)^l$ is a class-hiding space if and only if the DDH assumption holds in $\mathbb{G}_i$ [46].

### D. Structure-Preserving Signatures on Equivalence Classes

A structure-preserving signatures scheme on equivalence classes ($\mathsf{SPS-EQ}$) [45], [46] over a message equivalence relation $\mathcal{R}_{\vec{M}}$ consists of the following PPT algorithms:

- $\mathsf{SPS\text{-}EQ.Setup}(1^\lambda) \rightarrow \mathcal{BL}$: On input a security parameter $1^\lambda$, outputs a bilinear group $\mathcal{BL}$.
- $\mathsf{SPS\text{-}EQ.KeyGen}(\mathcal{BL}, l) \rightarrow (sk, pk)$: On input the bilinear group $\mathcal{BL}$ and a message length $l$, outputs a signing key $sk$ and a public key $pk$.
- $\mathsf{SPS\text{-}EQ.Sign}(sk, \vec{M}) \rightarrow \sigma$: On input the signing key $sk$ and a message $\vec{M}$, outputs a signature $\sigma$ on the message $\vec{M}$.
- $\mathsf{SPS\text{-}EQ.ChgRep}(\vec{M}, \sigma, \mu, pk) \rightarrow (\vec{M}', \sigma')$: On input a representative $\vec{M}$ of an equivalence class $[\vec{M}]_{\mathcal{R}}$, a signature $\sigma$ on the message $\vec{M}$, a message converter $\mu \in \mathbb{Z}_p^*$, and the public key $pk$, outputs an updated message-signature pair $(\vec{M}', \sigma')$, where $\vec{M}' \in [\vec{M}]_{\mathcal{R}}$.
- $\mathsf{SPS\text{-}EQ.Verify}(\vec{M}, \sigma, pk) \rightarrow 0/1$: On input a representative $\vec{M}$ of an equivalence class $[\vec{M}]_{\mathcal{R}}$, a signature $\sigma$ on the message $\vec{M}$, and the public key $pk$, outputs 1 if $\sigma$ is valid on $\vec{M}$ under $pk$ and 0 otherwise.
- $\mathsf{SPS\text{-}EQ.VKey}(sk, pk) \rightarrow 0/1$: On input the signing key $sk$ and the public key $pk$, outputs 1 if it is a valid key pair and 0 otherwise.

An $\mathsf{SPS-EQ}$ [46] scheme is secure if it is existentially unforgeable under chosen message attacks (EUF-CMA) and has perfect adaptation of signatures. We introduce the security definition of $\mathsf{SPS-EQ}$ and a concrete instantiation of it in Supplemental Material D.1, available online. In this paper, we use this signature scheme to hide issuers' public keys and credentials.

### E. Aggregatable Attribute-Based on Equivalence Classes

An aggregatable attribute-based equivalence class ($\mathsf{AAEQ}$) signature scheme over a message equivalence relation $\mathcal{R}_{\vec{M}}$ consists of the following PPT algorithms:

- $\mathsf{AAEQ.Setup}(1^\lambda, t, n) \rightarrow pp$: On input a security parameter $1^\lambda$, a message length $t$, and a total number of user's attributes $n$, outputs public parameters $pp$.
- $\mathsf{AAEQ.KeyGen}(pp) \rightarrow (sk, pk)$: On input the public parameters $pp$, outputs a signing key $sk$ and a public key $pk$.
- $\mathsf{AAEQ.Sign}(sk, a_j, \vec{M}) \rightarrow \sigma_j$: On input the signing key $sk$, an attribute value $a_j$, and a message $\vec{M}$, outputs a signature $\sigma$ of the attribute $a_j$ on the message $\vec{M}$.
- $\mathsf{AAEQ.Aggr}(pk, \{a_j, \sigma_j\}_{\mathcal{D}}) \rightarrow \sigma$: On input the public key $pk$ and a set of valid attribute-signature pairs $\{a_j, \sigma_j\}_{j \in \mathcal{D}}$ on the message $\vec{M}$, ($\mathcal{D} \subset [1, n]$), outputs an aggregated signature $\sigma$ for the set of attributes $\{a_j\}_{j \in \mathcal{D}}$.
- $\mathsf{AAEQ.ChgRep}(\vec{M}, \sigma, \rho, pk) \rightarrow (\vec{M}', \sigma')$: On input a message $\vec{M}$, a signature $\sigma$ and a message converter $\rho \in \mathbb{Z}_p^*$, outputs an updated message-signature pair $(\vec{M}', \sigma')$, where $\vec{M}' \in [\vec{M}]_{\mathcal{R}}$.
- $\mathsf{AAEQ.Verify}(pk, \{a_j\}_{\mathcal{D}}, \sigma, \vec{M}) \rightarrow 0/1$: On input a public key $pk$, a set of attributes $\{a_j\}_{j \in \mathcal{D}}$, and a message-signature pair $(\vec{M}, \sigma)$, outputs 1 if $\sigma$ is valid for $\{a_j\}_{j \in \mathcal{D}}$ and $\vec{M}$, and 0 otherwise.
- $\mathsf{AAEQ.VKey}(sk, pk) \rightarrow 0/1$: On input the signing key $sk$ and the public key $pk$, outputs 1 if it is a valid key pair and 0 otherwise.

An $\mathsf{AAEQ}$ [47] scheme is secure if it is EUF-CMA secure and has perfect adaptation of signatures. We introduce the security definition of $\mathsf{AAEQ}$ and a concrete instantiation of it in Supplemental Material D.2, available online. In this paper, we use this signature scheme as a building block to construct our $\mathsf{TAM-Sign}$ scheme.

### F. Updatable Public Keys

An updatable public key system ($\mathsf{UPK}$) over a public-key equivalence relation $\mathcal{R}_{pk}$ consists of the following PPT algorithms:

- $\mathsf{UPK.Setup}(1^\lambda) \rightarrow pp$: On input a security parameter $1^\lambda$, outputs the parameters $pp$.
- $\mathsf{UPK.KeyGen}(pp) \rightarrow (sk, pk)$: On input the public parameters $pp$, outputs a secret key $sk$ and a public key $pk$.
- $\mathsf{UPK.Update}(pk) \rightarrow pk'$: On input a public key $pk$, outputs a updated public key $pk'$, where $pk' \in [pk]_{\mathcal{R}}$..
- $\mathsf{UPK.VfyKey}(sk, pk) \rightarrow 0/1$: On input a secret key $sk$ and the public key $pk$, outputs 1 if it is a valid key pair and 0 otherwise.

A $\mathsf{UPK}$ [48] scheme is secure if it satisfies unforgeability and indistinguishability. We introduce the security definition of $\mathsf{UPK}$ and a concrete instantiation of it in Supplemental Material D.3, available online. In this paper, we use this $\mathsf{UPK}$ scheme as a building block to construct our $\mathsf{TAM-Sign}$ scheme.

### G. Zero-Knowledge Signature of Knowledge

Zero-knowledge signature of knowledge ($\mathsf{ZKSoK}$) [23] for a NP-relation $\mathcal{R}$ with the language $L_{\mathcal{R}} = \{y : \exists x, (x, y) \in \mathcal{R}\}$ consists of the following algorithms.

- $\mathsf{Gen}(1^\lambda) \to pp$*:* On input a security parameter $\lambda$, outputs a public parameter $pp$.
- $\mathsf{Sign}(x, y, m) \to \Pi$*:* On input a message $m$ and a relation $(x, y) \in \mathcal{R}$, outputs a ZKSoK: $\Pi = \mathsf{ZKSoK}\{x|(x, y) \in \mathcal{R}\}(m)$.
- $\mathsf{Verify}(y, \Pi, m) \to 0/1$*:* On input a message $m$, a ZKSoK $\Pi$ and a statement $y$. If $\Pi$ is valid, return 1; otherwise return 0.

A ZKSoK is $SimExt\text{-}secure$ [23] if it satisfies correctness, simulatability, and extractability. We introduce the security definition of ZKSoK and an instantiation approach in Supplemental Material D.4, available online.

## III. Tag-Based Aggregatable Mercurial Signatures

In this section, we introduce a new primitive called tag-based aggregatable mercurial signatures ($\mathsf{TAM-Sign}$). In this scheme, there are two equivalence classes: tag equivalence class $[\vec{T}]_{\mathcal{R}}$ and public-key equivalence class $[pk]_{\mathcal{R}}$, which are defined based on equivalence relations $\mathcal{R}_{\vec{T}}$ and $\mathcal{R}_{pk}$, respectively.

$\mathcal{R}_{\vec{T}} = \{(\vec{T}, \vec{T}') \in (\mathbb{G}_i^*)^t \times (\mathbb{G}_i^*)^t | \exists s \in \mathbb{Z}_p^* : \vec{T}' = \vec{T}^s\}$,

$\mathcal{R}_{pk} = \{(pk, pk') \in (\mathbb{G}_{i'}^*)^l \times (\mathbb{G}_{i'}^*)^l | \exists s \in \mathbb{Z}_p^* : pk' = pk^s\}$.

In $\mathsf{TAM-Sign}$ scheme, each user generates a tag $\vec{T}$ and corresponding witness; there is a signing key that can issue a signature for per attribute on a tag under a public key. This scheme is required to be aggregatable so that signatures for multiple attributes on the same representative $\vec{T}$ from a tag equivalence class can be aggregated into a compact signature. Moreover, the scheme allows a signature for the attributes on a tag under a public key to be randomized and converted to an unlinkable signature for the same attribute on an equivalent but unlinkable tag under an equivalent but unlinkable public key.

### A. Formal Definitions

A $\mathsf{TAM-Sign}$ scheme over equivalence relations $\mathcal{R}_{\vec{T}}$ and $\mathcal{R}_{pk}$ consists of the following PPT algorithms:

- $\mathsf{TAM\text{-}Sign.Setup}(1^\lambda, t, n) \to pp$*:* On input a security parameter $1^\lambda$, a tag length $t$ and a total number of user's attributes $n$, this probabilistic algorithm outputs the public parameters $pp$.
- $\mathsf{TAM\text{-}Sign.KeyGen}(pp) \to (sk, pk)$*:* On input the public parameters $pp$, this probabilistic algorithm outputs a signing key $sk$ and a public key $pk$.
- $\mathsf{TAM\text{-}Sign.GenTag}(pp) \to (tw, \vec{T})$*:* On input the public parameters $pp$, this probabilistic algorithm outputs a tag $\vec{T} \in (\mathbb{G}_i^*)^t$ and corresponding secret witness $tw$.
- $\mathsf{TAM\text{-}Sign.Sign}(sk, \vec{T}, a_j) \to \sigma_j$*:* On input the signing key $sk$, a tag $\vec{T}$ and an attribute value $a_j$ ($j \in [1, n]$), this probabilistic algorithm outputs a signature $\sigma_j$ of the attribute $a_j$ for the tag $\vec{T}$.
- $\mathsf{TAM\text{-}Sign.Aggr}(pk, \{a_j, \sigma_j\}_{j\in\mathcal{D}}, \vec{T}) \to \sigma/\perp$*:* On input the public key $pk$ and a set of valid attribute-signature pairs $\{a_j, \sigma_j\}_{j\in\mathcal{D}}$ on the same tag $\vec{T}$ ($\mathcal{D} \subset [1, n]$), this deterministic algorithm outputs an aggregated signature $\sigma$ of the set of attributes $\{a_j\}_{j\in\mathcal{D}}$ if it executes successfully, and $\perp$ otherwise.
- $\mathsf{TAM\text{-}Sign.ConvertSK}(sk, \mu) \to sk'$*:* On input the signing key $sk$ and a key converter $\mu \in \mathbb{Z}_p^*$, this deterministic algorithm outputs an updated signing key $sk'$.
- $\mathsf{TAM\text{-}Sign.ConvertPK}(pk, \mu) \to pk'$*:* On input the public key $pk$ and a key converter $\mu \in \mathbb{Z}_p^*$, this deterministic algorithm outputs an updated public key $pk'$ that satisfies $pk' \in [pk]_{\mathcal{R}}$.
- $\mathsf{TAM\text{-}Sign.ConvertSign}(\sigma, \mu) \to \sigma'$*:* On input the public key $pk$ and a key converter $\mu \in \mathbb{Z}_p^*$, this probabilistic algorithm outputs an updated signature $\sigma'$.
- $\mathsf{TAM\text{-}Sign.ChgRep}(\sigma, \vec{T}, \rho) \to (\vec{T}', \sigma')$*:* On input a signature $\sigma$, a tag $\vec{T}$ and a tag converter $\rho \in \mathbb{Z}_p^*$, this probabilistic algorithm outputs an updated tag-signature pair $(\vec{T}', \sigma')$, where $\vec{T}' \in [\vec{T}]_{\mathcal{R}}$.
- $\mathsf{TAM\text{-}Sign.Verify}(pk, \vec{T}, \sigma, \{a_j\}_{j\in\mathcal{D}}) \to 0/1$*:* On input a public key $pk$, a tag-signature pair $(\vec{T}, \sigma)$ and a set of attributes $\{a_j\}_{j\in\mathcal{D}}$, this deterministic algorithm outputs 1 if $\sigma$ is valid for $\{a_j\}_{j\in\mathcal{D}}$ and $\vec{T}$ under $pk$, and 0 otherwise.
- $\mathsf{TAM\text{-}Sign.VfyKey}(pk, sk) \to 0/1$*:* On input a public key $pk$ and a signing key $sk$, this deterministic algorithm checks the consistency of key pair and outputs 1 if successful, and 0 otherwise.
- $\mathsf{TAM\text{-}Sign.VfyTag}(tw, \vec{T}) \to 0/1$*:* On input a tag $\vec{T}$ and a witness $tw$, this deterministic algorithm checks the consistency of witness-tag pair and outputs 1 if successful, and 0 otherwise.

Given a set of attribute-signatures $\{a_j, \sigma_j\}_{j\in\mathcal{D}}$ on the same tag $\vec{T}$ under the public key $pk$, we can generate an aggregated signature $\sigma$ on the set of attributes $\{a_j\}_{j\in\mathcal{D}}$ and converter it to a new signature $\sigma'$ under a new equivalent public key $pk'$ without knowing the corresponding signing key. $\mathsf{TAM-Sign.ConvertSK}$, $\mathsf{TAM-Sign.ConvertPK}$ and $\mathsf{TAM-Sign.ConvertSign}$ can be seen as a matching randomization of the signing key pair and signature using a random key converter $\mu$ without invalidating the signature under the new public key. Moreover, the tag-signature pair $(\vec{T}, \sigma)$ under the public key $pk$ can be randomized to a new signature $\sigma'$ with a new equivalent tag $\vec{T}'$ without knowing the signing key. $\mathsf{TAM-Sign.ChgRep}$ can be seen as a matching randomization of a tag and signature using a random tag converter $\rho$ without invalidating the signature with the new tag.

*Correctness:* A $\mathsf{TAM-Sign}$ scheme is correct if it satisfies: (1) the verification of aggregated signatures is correct; (2) the conversion of keys and signatures is correct; (3) the change of tag representative is correct. Correct verification of aggregated signatures means that honestly generated keys, honestly generated tags, and honestly generated and aggregated signatures are always verified. Correct conversion of keys and signatures means that when a same key converter is applied to a signing key pair and a signature with a tag, it produces a valid updated signing key pair, where the updated and original public keys belong to the same equivalence class, and a valid updated signature with the tag under the new public key. Correctly changing the tag representative means that if a tag converter is applied to a tag-signature pair, then it obtains a valid updated tag-signature pair, where the updated and original tags belong to the same

$Exp^{uf_{\mathsf{TAM\text{-}Sign}}}(\mathcal{A}, 1^\lambda, t, n)$:
1. $pp \leftarrow \mathsf{TAM\text{-}Sign.Setup}(1^\lambda, t, n)$; $\mathcal{Q} = \emptyset$.
2. $(sk, pk) \leftarrow \mathsf{TAM\text{-}Sign.KeyGen}(pp)$.
3. $(pk^*, \vec{T}^*, \sigma^*, \{a_j^*\}_{j \in \mathcal{D}^*}) \leftarrow \mathcal{A}^{\mathcal{O}_{Sign}(sk,\cdot)}$.
4. If $\forall j \in \mathcal{D}^*, (a_j^*, [\vec{T}^*]_\mathcal{R}) \notin \mathcal{Q}$, $pk^* \in [pk]_\mathcal{R}$, and $\mathsf{TAM\text{-}Sign.Verify}(pk^*, \{a_j^*\}_{j \in \mathcal{D}^*}, \sigma^*, \vec{M}^*) = 1$, return 1.

Fig. 1. EUF-CMA of TAM-sign.

equivalence class. The formal correctness definition is shown in Supplemental Material B.1, available online.

### B. Security Definitions

The security definition of EUF-CMA requires the following global variable and oracle.

– $\mathcal{Q}$: A set of signed attribute-tag pairs $(a_j, \vec{T})$.

– $\mathcal{O}_{Sign}(sk, \vec{T}, a_j)$: It is an oracle that can be used to issue a signature for an attribute $a_j$ on a tag $\vec{T}$. It runs $\sigma \leftarrow \mathsf{TAM\text{-}Sign.Sign}(sk, \vec{T}, a_j)$, adds $(a_j, \vec{T})$ to $\mathcal{Q}$ ($\mathcal{Q} := \mathcal{Q} \cup (a_j, \vec{T})$), and returns $\sigma_j$.

*Definition III.1:* The EUF-CMA is defined by experiment $Exp^{uf_{\mathsf{TAM\text{-}Sign}}}$ in Fig. 1. A TAM-Sign scheme is EUF-CMA secure if for $t \in \mathbb{Z}^+$, $n \in \mathbb{Z}^+$ and any PPT adversary $\mathcal{A}$ having access to the oracle $\mathcal{O}_{Sign}(sk, \cdot)$, there is a negligible function $\epsilon(\lambda)$ such that:

$$Adv^{uf_{\mathsf{TAM\text{-}Sign}}} = \left|\Pr\left[Exp^{uf_{\mathsf{TAM\text{-}Sign}}}(\mathcal{A}, 1^\lambda, t, n)\right] = 1\right| \leqslant \epsilon(\lambda)$$

Compared with the definition of standard signature's EUF-CMA, the unforgeability of TAM-Sign is different in the following two aspects, except that the signature output by the adversary requires correct verification.

(1) The adversary's forgery is invalid if there is attribute-tag pair $\{a_j^*, \vec{T}^*\}$ belonging to the attributes and tag equivalence classes of the previous query, i.e. $\forall j \in \mathcal{D}, (a_j^*, [\vec{T}^*]_\mathcal{R}) \notin \mathcal{Q}$.

(2) The adversary is allowed to forge a public key $pk^*$ as long as it belongs to the same equivalence class as the original public key $pk$, i.e. $pk^* \in [pk]_\mathcal{R}$.

*Definition III.2. Signature Adaptation:* An $\mathsf{TAM-Sign}$ scheme perfectly adapts signatures if for all tuples $(sk, pk, tw, \vec{T}, \{a_j\}, \sigma, \mu, \rho)$, the following conditions are met:

1) $\mathsf{TAM\text{-}Sign.VfyKey}(pk, sk) = 1$.
2) $\mathsf{TAM\text{-}Sign.VfyTag}(tw, \vec{T}) = 1$.
3) $\mathsf{TAM\text{-}Sign.Verify}(pk, \vec{T}, \sigma, \{a_j\}) = 1$.
4) $(\mu, \rho) \in \mathbb{Z}_p^*$, $\vec{T} \in (\mathbb{G}_i^*)^t$, $pk \in (\mathbb{G}_{i'}^*)^l$, where $l = f(t, n) \in \mathbb{Z}^+$.

$(pk^\mu, \vec{T}^\rho, \mathsf{TAM\text{-}Sign.Aggr}(pk^\mu, \{a_j, \mathsf{TAM\text{-}Sign.Sign}(sk \cdot \mu, \vec{T}^\rho, a_j)\}))$ and $(\mathsf{TAM\text{-}Sign.ConvertPK}(pk, \mu), \mathsf{TAM\text{-}Sign.ChgRep}(\mathsf{TAM\text{-}Sign.ConvertSign}(\sigma, \mu), \vec{T}, \rho))$ are identically distributed.

$Exp^{uf_{\mathsf{Tag}}}(\mathcal{A}, 1^\lambda, t, n)$:
1. $pp \leftarrow \mathsf{TAM\text{-}Sign.Setup}(1^\lambda, t, n)$.
2. $(tw, \vec{T}) \leftarrow \mathsf{TAM\text{-}Sign.GenTag}(pp)$.
3. $(tw^*, \vec{T}^*) \leftarrow \mathcal{A}(pp, \vec{T})$.
4. If $\vec{T}^* \in [\vec{T}]_\mathcal{R}$, and $\mathsf{TAM\text{-}Sign.VfyTag}(tw^*, \vec{T}^*) = 1$, return 1.

Fig. 2. Unforgeability of tag.

We follow the idea of Fuchsbauer [46] and Hanzlik [47] in defining signature adaptation. Signature adaptation is a formalization of two notions. First, the $\mathsf{TAM-Sign}$ signature $(pk, \vec{T}, \sigma)$ parameterized by equivalence relations $\mathcal{R}_{\vec{T}}$ and $\mathcal{R}_{pk}$ is class-hiding. Second, any change in the representation of a valid signature is distributed as a new signature. This ensures user protection against the signer, especially when randomizing tag-signature pairs received from the signer.

*Definition III.3. Unforgeability of Tags:* The unforgeability of tags is defined by experiment $Exp^{uf_{\mathsf{Tag}}}$ in Fig. 2. A TAM-Sign scheme satisfies unforgeability of tags if for any PPT adversary $\mathcal{A}$, there is a negligible function $\epsilon(\lambda)$ such that:

$$Adv^{uf_{\mathsf{Tag}}} = \left|\Pr\left[Exp^{uf_{\mathsf{Tag}}}(\mathcal{A}, 1^\lambda, t, n)\right] = 1\right| \leqslant \epsilon(\lambda)$$

Unforgeability of tags ensures that an adversary should not be able to learn the secret witness of an updated tag unless it already knew the witness for the original tag.

### C. Concrete Instantiation

Our $\mathsf{TAM-Sign}$ scheme is inspired by the aggregatable attribute-based equivalence class signature (AAEQ) [47], updatable public keys scheme [48], and mercurial signatures [33]. The construction of the $\mathsf{TAM-Sign}$ scheme follows a three-step process. (1) Employ updatable public key as user tag, which allows users to control the presentation of their signatures for verification. (2) We replace the messages in AAEQ scheme with user-generated and updatable tag. Multiple signatures on the same tag can be aggregated into a compact signature. (3) Following the paradigm of mercurial signatures, we allow the signature on a tag under a public key to be randomized and converted to a new unlinkable signature on an equivalent but unlinkable tag under an equivalent but unlinkable public key.

In this section, we present a concrete instantiation of $\mathsf{TAM-Sign}$, which combines the AAEQ [47] scheme of Hanzlik et al. the updatable public key (UPK) [48] scheme of Fauzi et al. and the mercurial signatures [33] of Crites et al.

– $\mathsf{TAM\text{-}Sign.Setup}(1^\lambda, t = 2, n) \rightarrow pp$:

- Generate a Type-III bilinear group $\mathcal{BL} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, p, g, \widetilde{g})$.
- Choose a pseudorandom function PRF and a collision resistant hash function HASH: $\{0,1\}^* \rightarrow \mathbb{G}_1^*$.
- Set the equivalence classes $[\vec{T}]_\mathcal{R} \subset (\mathbb{G}_1^*)^2$ and $[pk]_\mathcal{R} \subset (\mathbb{G}_2^*)^{2 \cdot n}$.
- Set $pp = (t{=}2, n, \mathcal{BL}, \mathsf{PRF}, \mathsf{HASH}, [\vec{T}]_\mathcal{R}, [pk]_\mathcal{R})$.

– $\mathsf{TAM\text{-}Sign.KeyGen}(pp) \rightarrow (sk, pk)$:

- Compute $rk \leftarrow \mathsf{PRF.KeyGen}(1^\lambda)$.

- For all $j \in [1,n]$, choose $(x_{j,1}, x_{j,2}) \xleftarrow{R} \mathbb{Z}_p^*$ and set $sk_j = (x_{j,1}, x_{j,2})$.
- Compute $pk_j = (\widetilde{X}_{j,1}, \widetilde{X}_{j,2}) \leftarrow (\widetilde{g}^{x_{j,1}}, \widetilde{g}^{x_{j,2}})$.
- Set $sk = (rk, sk_1, \ldots, sk_n)$ and $pk = (pk_1, \ldots, pk_n)$.

– $\mathsf{TAM\text{-}Sign.GenTag}(pp) \rightarrow (tw, \vec{T})$.
- Choose $(tw, r) \xleftarrow{R} \mathbb{Z}_p^*$ and compute $\vec{T} = (T_1, T_2) \leftarrow (g^r, g^{r \cdot tw})$.

– $\mathsf{TAM\text{-}Sign.Sign}(sk, \vec{T}, a_j) \rightarrow \sigma_j$.
- Check $a_j \in \{0,1\}^*$ and $\vec{T} \in (\mathbb{G}_1^*)^2$.
- Generate a random $\gamma = \mathsf{PRF.Eval}(rk, \vec{T})$ and $\gamma \in \mathbb{Z}_p^*$.
- Compute $\sigma_j = (Z_j, Y_j, \widetilde{Y}_j, V_j) \leftarrow ((\prod_{i=1}^2 T_i^{x_{j,i}})^\gamma, g^{1/\gamma}, \widetilde{g}^{1/\gamma}, \mathsf{HASH}(a_j)^{1/\gamma})$.

– $\mathsf{TAM\text{-}Sign.Aggr}(pk, \{a_j, \sigma_j\}_{j \in \mathcal{D}}, \vec{T}) \rightarrow \sigma / \perp$.
- If $\mathcal{D} \nsubseteq [1,n]$, output $\perp$.
- Compute $\sigma = (Z, Y, \widetilde{Y}, V) \leftarrow (\prod_{j \in \mathcal{D}} Z_j, Y_{j^*}, \widetilde{Y}_{j^*}, \prod_{j \in \mathcal{D}} V_j)$, where $j^* \in \mathcal{D}$.

– $\mathsf{TAM\text{-}Sign.ConvertSK}(sk, \mu) \rightarrow sk'$.
- Compute $sk' = (rk, sk_1 \cdot \mu, \ldots, sk_n \cdot \mu)$, where $sk_j \cdot \mu = (x_{j,1} \cdot \mu, x_{j,2} \cdot \mu)$.

– $\mathsf{TAM\text{-}Sign.ConvertPK}(pk, \mu) \rightarrow pk'$.
- Compute $pk' = pk^\mu = (pk_1^\mu, \ldots, pk_n^\mu)$, where $pk_j^\mu = (\widetilde{X}_{j,1}^\mu, \widetilde{X}_{j,2}^\mu)$.

– $\mathsf{TAM\text{-}Sign.ConvertSign}(\sigma, \mu) \rightarrow \sigma'$.
- Choose $\phi_1 \xleftarrow{R} \mathbb{Z}_p^*$ and compute $\sigma' = (Z', Y', \widetilde{Y}', V') \leftarrow (Z^{\phi_1 \cdot \mu}, Y^{1/\phi_1}, \widetilde{Y}^{1/\phi_1}, V^{1/\phi_1})$.

– $\mathsf{TAM\text{-}Sign.ChgRep}(\sigma, \vec{T}, \rho) \rightarrow (\vec{T}', \sigma')$.
- Compute $\vec{T}' = \vec{T}^\rho \leftarrow (T_1^\rho, T_2^\rho)$.
- Choose $\phi_2 \xleftarrow{R} \mathbb{Z}_p^*$ and compute $\sigma' = (Z', Y', \widetilde{Y}', V') \leftarrow (Z^{\phi_2 \cdot \rho}, Y^{1/\phi_2}, \widetilde{Y}^{1/\phi_2}, V^{1/\phi_2})$.

– $\mathsf{TAM\text{-}Sign.Verify}(pk, \vec{T}, \sigma, \{a_j\}_{j \in \mathcal{D}}) \rightarrow 0/1$.
- If $\mathcal{D} \nsubseteq [1,n]$, output 0.
- Check whether $\prod_{i=1}^2 e(T_i, \prod_{j \in \mathcal{D}} \widetilde{X}_{j,i}) = e(Z, \widetilde{Y})$.
- Check whether $e(Y, \widetilde{g}) = e(g, \widetilde{Y})$.
- Check whether $e(\prod_{j \in \mathcal{D}} \mathsf{HASH}(a_j), \widetilde{Y}) = e(V, \widetilde{g})$.
- Output 1 if the above checks hold and 0 otherwise.

– $\mathsf{TAM\text{-}Sign.VfyKey}(pk, sk) \rightarrow 0/1$.
- For all $i \in [1,2]$ and $j \in [1,n]$, check whether $\widetilde{X}_{j,i} = \widetilde{g}^{x_{j,i}}$.
- Output 1 if the above checks hold and 0 otherwise.

– $\mathsf{TAM\text{-}Sign.VfyTag}(tw, \vec{T}) \rightarrow 0/1$.
- If $T_2 = T_1^{tw}$, output 1, otherwise output 0.

### D. Security Analysis

In this section, we prove the security of $\mathsf{TAM-Sign}$ scheme.

*Theorem III.1:* Our $\mathsf{TAM-Sign}$ scheme is correct. If the AAEQ scheme and the UPK scheme satisfy correctness, then our $\mathsf{TAM-Sign}$ scheme is correct.

*Theorem III.2:* Our $\mathsf{TAM-Sign}$ scheme is unforgeable in the generic group model if the underlying AAEQ scheme is unforgeable.

*Proof:* To prove the EUF-CMA of $\mathsf{TAM-Sign}$, we construct a simulator $\mathcal{S}$ to attack the EUF-CMA of the AAEQ scheme. Suppose a PPT adversary $\mathcal{A}$ produces a successful forgery $(pk^*, \vec{T}^*, \sigma^*, \{a_j^*\}_{j \in \mathcal{D}^*})$ for the $\mathsf{TAM-Sign}$ scheme with non-negligible probability $\epsilon(\lambda)$. Then, by definition, there exists a $\mu^* \in \mathbb{Z}_p^*$ such that $pk^* = pk^{\mu^*}$, where $pk$ is the challenged public key for EUF-CMA of the AAEQ scheme. We show that $\mathcal{S}$ is able to recover the key converter $\mu^*$ and generate a successful forgery $((\vec{T}^*)^\mu, \sigma^*, \{a_j^*\}_{j \in \mathcal{D}^*})$ for the AAEQ scheme under the public key $pk$, contradicting its proven security in the generic group model. The detailed proof is given in Supplemental Material B.2.1, available online.

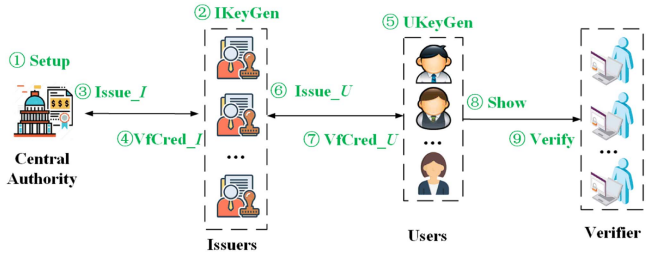



Fig. 3. Double issuer-hiding anonymous credentials.

*Theorem III.3:* Our $\mathsf{TAM-Sign}$ scheme provides perfect adaption of signatures.

We provide this proof in Supplemental Material B.2.2, available online.

*Theorem III.4:* Our $\mathsf{TAM-Sign}$ scheme satisfies unforgeability of tags.

*Proof:* This straightforwardly follows from the unforgeability of the underlying UPK scheme.

## IV. Double Issuer-Hiding Anonymous Credentials

### A. System Architecture

As shown in Fig. 3, the architecture of DIHAC scheme consists of four types of parties: a central authority (CA), issuers (I), users (U), and verifiers (V). The specific role of each party is described as follows.

- CA is a trusted global party responsible for setting up the system (step ①) and providing certification service for a group of issuers (step ③).
- I is an independent issuer and should register to CA (step ② and ④). There are multiple issuers in the DIHAC scheme whose task is to anonymously issue attribute-based credentials to users (step ⑥). In our system architecture, each issuer independently obtains its credential from CA. Consequently, even if one of the issuers is corrupted, the adverse impact on the system is limited.
- U with a set of attributes should apply for credentials from one of the authenticated issuers (step ⑤ and ⑦), but he can not determine the issuer's identity. Then, U anonymously presents a credential to V (step ⑧) while disclosing a subset of his attributes.
- V provides credential verification services for all users (step ⑨). V can obtain some of the attribute values that the user voluntarily discloses and verify that the user's credential was issued by an authenticated issuer, but it can neither determine the user's hidden attribute values nor issuer's identity.

### B. Formal Definitions

A DIHAC scheme consists of the following PPT algorithms:

- $\mathsf{DIHAC.Setup}(1^\lambda) \rightarrow (pp, msk, mpk)$. This algorithm is operated by the CA that inputs a security parameter $1^\lambda$, then outputs the system parameters $pp$ and a master private/public key pair $(msk, mpk)$.
- $\mathsf{DIHAC.IKeyGen}(pp) \rightarrow (isk, ipk, \Pi_1)$. This algorithm is operated by an issuer that inputs the system parameters $pp$, then outputs a secret key $isk$, a public key $ipk$, as well as a signature of knowledge $\Pi_1$ to demonstrate the issuer knows the secret key $isk$.
- $\mathsf{DIHAC.Issue}_I(msk, mpk, ipk, \Pi_1) \rightarrow icred/\perp$. This algorithm is operated by the CA that takes the master private/public key pair $(msk, mpk)$, an issuer's public key $ipk$ and corresponding signature of knowledge $\Pi_1$ as inputs. The algorithm outputs either a credential $icred$ for the issuer if it succeeds or $\perp$ if it fails.
- $\mathsf{DIHAC.VfCred}_I(mpk, isk, ipk, icred) \rightarrow 0/1$. This algorithm is operated by an issuer that takes the master public key $mpk$, the issuer's own key pair $(isk, spk)$, and a credential $icred$ as inputs. The algorithm outputs 1 if the credential is validated, 0 otherwise.
- $\mathsf{DIHAC.UKeyGen}(pp) \rightarrow (usk, upk, \Pi_2)$. This algorithm is operated by a user that inputs the system parameters $pp$, then outputs a secret key $usk$, a public key $upk$, as well as a signature of knowledge $\Pi_2$ to demonstrate the user knows the secret key $usk$.
- $\mathsf{DIHAC.Issue}_U(isk, ipk, icred, upk, \{a_j\}_{j\in[1,n]}, \Pi_2) \rightarrow (ipk', icred', ucred)/\perp$. This algorithm is operated by an issuer that takes the issuer's key pair $(isk, ipk)$, the issuer's credential $icred$, a set of user's attributes $\{a_j\}_{j\in[1,n]}$, a user's public key $upk$ and corresponding signature of knowledge $\Pi_2$ as inputs. If it succeeds, the algorithm outputs a credential $ucred$ for the set of attributes of the user and the corresponding unlinkable public key $ipk'$ and credential $icred'$ of an issuer, or $\perp$ if it fails.
- $\mathsf{DIHAC.VfCred}_U(mpk, ipk', icred', usk, upk, \{a_j\}_{j\in[1,n]}, ucred) \rightarrow 0/1$. This algorithm is operated by a user that takes the master public key $mpk$, an issuer's public key $ipk'$ and corresponding credential $icred'$, the user's own key pair $(usk, upk)$, and a user credential $ucred$ as inputs. The algorithm outputs 1 if the issuer and the user credentials are all validated, and 0 otherwise.
- $\mathsf{DIHAC.Show}(mpk, ipk', icred', usk, upk, ucred, \{a_j\}_{j\in\mathcal{D}}, \mathsf{CTX}) \rightarrow tk$. This algorithm is operated by a user that takes the master public key $mpk$, an issuer's public key $ipk'$ and corresponding credential $icred'$, the user's own key pair $(usk, upk)$ and corresponding credential $ucred$, a set of selective disclosure attributes $\{a_j\}_{j\in\mathcal{D}}$ ($\mathcal{D} \subset [1, n]$), and a context $\mathsf{CTX}$ as inputs, then outputs a presentation token $tk$. Note that $\{a_j\}_{j\in\mathcal{D}}$ should satisfy an attribute disclosure policy enforced by a verifier in order to pass the verifier's verification, and $\mathsf{CTX}$ includes a random message to prevent replay attacks.
- $\mathsf{DIHAC.Verify}(mpk, \{a_j\}_{j\in\mathcal{D}}, tk, \mathsf{CTX}) \rightarrow 0/1$. This algorithm is operated by a verifier that takes the master public key $mpk$, a set of user's attributes $\{a_j\}_{j\in\mathcal{D}}$, a presentation token $tk$ and a context $\mathsf{CTX}$ as inputs. The algorithm outputs 1 if the user's attributes satisfies an attribute disclosure policy enforced by the verifier and the token is validated, and 0 otherwise.

*Correctness:* A DIHAC scheme is correct if it satisfies: (1) the verification of issuer credentials is correct; (2) the verification of user credentials is correct, and (3) the verification of presentation tokens is correct. The formal definition of correctness is shown in Supplemental Material C.1, available online.

### C. Overflow of DIHAC

As shown in Fig. 3, the working flow of DIHAC scheme is described as follows. CA initializes the system and outputs system parameter $pp$ and a master private/public key pair $(msk, mpk)$ (DIHAC.Setup,step ①). To qualify to issue credentials for users, each issuer I generates a key pair $(isk, ipk)$ and the corresponding signature of knowledge $\Pi_1$ (DIHAC.IKeyGen, step ②); then, I authenticates itself to the CA ($\mathsf{DIHAC.Issue}_I$, step ③) and obtains its public key credential $icred$ ($\mathsf{DIHAC.VfCred}_I$, step ④). When joining the system, each user U generates a key pair $(usk, upk)$ and the corresponding signature of knowledge $\Pi_2$ (DIHAC.UKeyGen, step ⑤); then U authenticates himself to one of the authenticated issuers ($\mathsf{DIHAC.Issue}_U$, step ⑥) and obtains his attribute-based credential $ucred$ and the corresponding unlinkable public key $ipk'$ and unlinkable credential $icred'$ of the issuer ($\mathsf{DIHAC.VfCred}_U$, step ⑦). When showing a user credential to any verifier V, the user U anonymously proves the validity of the credential to V using a presentation token and reveals the set of disclosed attributes (DIHAC.Show, step ⑧), and V verifies the correctness of the presentation token (DIHAC.Verify, step ⑨).

### D. Threat Model

We assume the central authority CA is a fully trusted party in the system. Issuers I are assumed to be malicious. Unregistered malicious issuers may use fake credentials to issue attribute-based credentials for users, thereby stealing the identities of honest users or tricking them into failing authentication by the verifiers. Users U are assumed to be malicious because they may forge presentation tokens of credentials and attempt to obtain the identity information of issuers. Verifiers V are assumed to be honest but curious in the sense that it is honest in verifying the presentation tokens for users but is curious to obtain users' undisclosed attributes and the identity information of the issuers who issued users' credentials.

### E. Security Definitions

In this section, we define the security properties, including unforgeability, anonymity, and unlinkability of DIHAC. All security definitions use the following global variables and oracles. A challenger $\mathcal{C}$ is responsible for setting system parameters and generating keys for CA, all issuers, and all honest users. In addition, $\mathcal{C}$ is responsible for initializing and controlling all oracles and global variables. A PPT adversary $\mathcal{A}$ interacts with

$\mathcal{C}$, which simulates the behavior of honest participants (CA, all uncorrupted issuers, and all honest users) through the oracles.

*Global Variables*.

$\mathcal{Q}_{\mathsf{HU}}$: A set storing $(u, i, usk_u, upk_u, ucred_u, ipk'_i, icred'_i, \{a_j\}_{j\in[1,n]})$ each time an issuer $i$ issues credential for an honest user $u$.

$\mathcal{Q}_{\mathsf{CU}}$: A set containing the index $u$ of corrupt users.

$\mathcal{Q}_{\mathsf{HI}}$: A set storing $(i, isk, ipk)$ each time a credential is issued for an issuer $i$.

$\mathcal{Q}_{\mathsf{CI}}$: A set containing the index $i$ of corrupt issuers.

$\mathcal{Q}_{\mathsf{Shw}}$: A set storing $(\{a_j\}_{j\in\mathcal{D}}, \mathsf{CTX})$ each time a credential is shown to a verifier.

$\mathcal{Q}_{\mathsf{RI}}$: A set containing the revealed user-issuer index pair $(u, i)$.

*Oracles*.

– $\mathcal{O}_{\mathsf{Iss}_I}(i)$: It is an oracle that can be used to issue a public key credential for the issuer $i$. It generates a key pair $(isk_i, ipk_i)$ by running $(isk_i, ipk_i, \Pi_{1,i}) \leftarrow \mathsf{DIHAC.IKeyGen}(pp)$, and then issues a credential to issuer $i$ by running $icred_i \leftarrow \mathsf{DIHAC.Issue}_I(msk, mpk, ipk_i, \Pi_{1,i})$. Finally, it adds $(i, isk_i, ipk_i)$ to $\mathcal{Q}_{\mathsf{HI}}$ and returns $(i, ipk_i, icred_i)$.

– $\mathcal{O}_{\mathsf{Cor}_I}(i)$: It is an oracle that can be used to corrupt an issuer $i$. If $i \in \mathcal{Q}_{\mathsf{CI}}$, then it returns $\perp$. Otherwise, it deletes $(i, isk_i, ipk_i)$ from $\mathcal{Q}_{\mathsf{HI}}$ and adds $i$ to $\mathcal{Q}_{\mathsf{CI}}$. Finally, it returns $(i, isk_i)$.

– $\mathcal{O}_{\mathsf{ObtIss}_U}(u, \{a_j\}_{j\in[1,n]})$: It is an oracle that can be used to play an uncorrupted issuer issuing an attribute-based credential to an honest user $u$. If $u \in \mathcal{Q}_{\mathsf{CU}}$, it returns $\perp$. Otherwise, it randomly selects an issuer $i$ from $\mathcal{Q}_{\mathsf{HI}}$ and generates a key pair $(usk_u, upk_u)$ by running $(usk_u, upk_u, \Pi_{2,u}) \leftarrow \mathsf{DIHAC.UKeyGen}(pp)$; then it issues a credential to user $u$ by running $(ipk'_i, icred'_i, ucred_u) \leftarrow \mathsf{DIHAC.Issue}_U(isk_i, ipk_i, upk_u, \{a_j\}_{j\in[1,n]}, \Pi_{2,u})$. Finally, it adds $(u, i, usk_u, upk_u, ucred_u, ipk'_i, icred'_i, \{a_j\}_{j\in[1,n]})$ to $\mathcal{Q}_{\mathsf{HU}}$ and returns $(upk_u, ucred_u, ipk'_i, icred'_i)$.

– $\mathcal{O}_{\mathsf{Obt}_U}(u, \{a_j\}_{j\in[1,n]})$: It is an oracle that can be used to play a corrupted issuer issuing an attribute-based credential to an honest user $u$. If $u \in \mathcal{Q}_{\mathsf{CU}}$, it returns $\perp$. Otherwise, it randomly selects an issuer $i$ from $\mathcal{Q}_{\mathsf{CI}}$ and generates a key pair $(usk_u, upk_u)$ for user $u$ by running $(usk_u, upk_u, \Pi_{2,u}) \leftarrow \mathsf{DIHAC.UKeyGen}(pp)$, and then request a credential for user $u$ by querying adversary: $(ipk'_i, icred'_i, ucred_u) \leftarrow \mathcal{A}(isk_i, ipk_i, icred_i, upk_u, \{a_j\}_{j\in[1,n]}, \Pi_{2,u})$. Finally, it adds $(u, i, usk_u, upk_u, ucred_u, ipk'_i, icred'_i, \{a_j\}_{j\in[1,n]})$ to $\mathcal{Q}_{\mathsf{HU}}$ and returns 0.

– $\mathcal{O}_{\mathsf{Cor}_U}(u)$: It is an oracle that can be used to corrupt an honest user $u$. If $u \in \mathcal{Q}_{\mathsf{CU}}$, then it returns $\perp$. If $u \in \mathcal{Q}_{\mathsf{HU}}$, then it removes $(u, i, usk_u, upk_u, ucred_u, ipk'_i, icred'_i, \{a_j\}_{j\in[1,n]})$ from $\mathcal{Q}_{\mathsf{HU}}$ and adds $u$ to $\mathcal{Q}_{\mathsf{CU}}$. Finally, it returns $(i, usk_u)$.

– $\mathcal{O}_{\mathsf{Iss}_U}(u, upk_u, \Pi_{2,u}, \{a_j\}_{j\in[1,n]})$: It is an oracle that can be used to play an uncorrupted issuer issuing an attribute-based credential to a corrupt user $u$ with a public key $upk_u$ and a set of attributes $\{a_j\}_{j\in[1,n]}$. If $u \in \mathcal{Q}_{\mathsf{HU}} \cup \mathcal{Q}_{\mathsf{CU}}$, it returns $\perp$. Otherwise, it randomly selects an issuer $i$ from $\mathcal{Q}_{\mathsf{HI}}$ and issues a credential to user $u$ by running $(ipk'_i, icred'_i, ucred_u) \leftarrow \mathsf{DIHAC.Issue}_U(isk_i, ipk_i, upk_u, \{a_j\}_{j\in[1,n]}, \Pi_{2,u})$. Finally, it adds $u$ to $\mathcal{Q}_{\mathsf{CU}}$ and returns $(ipk'_i, icred'_i, ucred_u)$.

$Exp^{uf_{\mathsf{DIHAC}}}(\mathcal{A}, \lambda)$:
1. $(pp, msk, mpk) \leftarrow \mathsf{DIHAC.Setup}(1^\lambda)$;
2. $\mathcal{Q}_{\mathsf{HU}} \leftarrow \emptyset, \mathcal{Q}_{\mathsf{CU}} \leftarrow \emptyset, \mathcal{Q}_{\mathsf{HI}} \leftarrow \emptyset, \mathcal{Q}_{\mathsf{CI}} \leftarrow \emptyset,$ $\mathcal{Q}_{\mathsf{Shw}} \leftarrow \emptyset, \mathcal{Q}_{\mathsf{RI}} \leftarrow \emptyset$;
3. $(u^*, tk^*, \{a_j^*\}_{j\in\mathcal{D}^*}, \mathsf{CTX}^*) \leftarrow \mathcal{A}^{\mathcal{O}}(pp, mpk)$;
4. If $u^* \in \mathcal{Q}_{\mathsf{CU}}$, return 0;
5. If $\mathsf{DIHAC.Verify}(mpk, \{a_j^*\}_{j\in\mathcal{D}^*}, tk^*, \mathsf{CTX}^*) = 0$, return 0;
6. If $(\{a_j^*\}_{j\in\mathcal{D}^*}, \mathsf{CTX}^*) \in \mathcal{Q}_{\mathsf{Shw}}$, return 0;
7. Return 1.

Fig. 4. Unforgeability of DIHAC.

– $\mathcal{O}_{\mathsf{Shw}}(u, \mathcal{D}, \mathsf{CTX})$: It is an oracle that can be used to play an honest user $u$ with a set of selective attributes $\{a_j\}_{j\in\mathcal{D}}$ presenting a token of the credential to an honest but curious verifier. If $u \notin \mathcal{Q}_{\mathsf{HU}}$, it returns $\perp$. Otherwise, it generates a token by running $tk \leftarrow \mathsf{DIHAC.Show}(mpk, ipk'_i, icred'_i, usk_u, upk_u, ucred_u, \{a_j\}_{j\in\mathcal{D}}, \mathsf{CTX})$. Finally, it adds $(\{a_j\}_{j\in\mathcal{D}}, \mathsf{CTX})$ to $\mathcal{Q}_{\mathsf{Shw}}$ and returns $tk$.

– $\mathcal{O}_{\mathsf{RelIss}}(u)$: It is an oracle that can be used to reveal the issuer's index of the user $u$'s credential. It searches the issuer's index $i$ corresponding to user's index $u$ from $\mathcal{Q}_{\mathsf{HU}}$, adds $(u, i)$ to $\mathcal{Q}_{\mathsf{RI}}$ and returns $i$.

– $\mathcal{O}_{\mathsf{AnCh}_b}(u, upk_u, \Pi_{2,u}, i_0, i_1, \{a_j\}_{j\in[1,n]})$: It is a challenge oracle in the anonymity experiment where the adversary playing as a user is challenged to distinguish credentials issued by two different issuers. It takes a user $u$ with a public key $upk_u$ and a set of user's attributes $\{a_j\}_{j\in[1,n]}$, and two indexes of issuers $(i_0, i_1)$ as inputs. It runs: $(ipk'_{i_b}, icred'_{i_b}, ucred_u) \leftarrow \mathsf{DIHAC.Issue}_U(isk_{i_b}, ipk_{i_b}, icred_{i_b}, upk_u, \{a_j\}_{j\in[1,n]}, \Pi_{2,u})$, and returns $(upk_u, ucred_u, ipk'_{i_b}, icred'_{i_b})$, where $b = \{0, 1\}$.

– $\mathcal{O}_{\mathsf{UnCh}_b}(\{u_0, i_0\}, \{u_1, i_1\}, \mathcal{D}, \mathsf{CTX})$: It is a challenge oracle in the unlinkability experiment where the adversary playing as a verifier is challenged to distinguish tokens of two credentials issued by two different issuers. It takes the indexes of two user-issuer $\{u_0, i_0\}, \{u_1, i_1\}$, a selective disclosure policy $\mathcal{D}$, and a context $\mathsf{CTX}$ as inputs. It runs: $tk_b \leftarrow \mathsf{DIHAC.Show}(mpk, ipk'_{i_b}, icred'_{i_b}, usk_{u_b}, upk_{u_b}, ucred_{u_b}, \{a_j^{u_b}\}_{j\in\mathcal{D}}, \mathsf{CTX})$, and returns $tk_b$, where $b = \{0, 1\}$.

*Unforgeability:* Unforgeability protects honest verifiers from malicious users. In the unforgeability experiment, suppose an adversary playing as a malicious user has not previously received a credential on a set of disclosed attributes $\{a_j\}_{j\in\mathcal{D}}$ from one of the authenticated issuers. In that case, it is infeasible for the adversary to generate a valid token that satisfies the selective disclosure policy $\mathcal{D}$.

In the following definition, we note that while all the issuers' keys are generated by the challenger, the adversary can control the total number of issuers in the security games by querying the oracle $\mathcal{O}_{\mathsf{Iss}_I}$, and may request any user's credentials by querying the oracles $\mathcal{O}_{\mathsf{ObtIss}_U}$ and $\mathcal{O}_{\mathsf{Iss}_U}$.

*Definition IV.1. Unforgeability:* The unforgeability is defined by experiment $Exp^{uf_{\mathsf{DIHAC}}}$ in Fig. 4. Users'

$Exp_b^{ano_{\mathsf{DIHAC}}}(\mathcal{A}, \lambda)$:
1. $(pp, msk, mpk) \leftarrow \mathsf{DIHAC.Setup}(1^\lambda)$;
2. $\mathcal{Q}_{\mathsf{HU}} \leftarrow \emptyset, \mathcal{Q}_{\mathsf{CU}} \leftarrow \emptyset, \mathcal{Q}_{\mathsf{HI}} \leftarrow \emptyset, \mathcal{Q}_{\mathsf{CI}} \leftarrow \emptyset, \mathcal{Q}_{\mathsf{Shw}} \leftarrow \emptyset, \mathcal{Q}_{\mathsf{RI}} \leftarrow \emptyset$;
3. $(u, upk_u, \Pi_{2,u}, i_0, i_1, \{a_j\}_{j\in[1,n]}, st) \leftarrow \mathcal{A}^{\mathcal{O}}(pp, mpk)$;
4. If $u \in \mathcal{Q}_{\mathsf{HU}} \cup \mathcal{Q}_{\mathsf{CU}}$ or $i_0 \notin \mathcal{Q}_{\mathsf{HI}}$ or $i_1 \notin \mathcal{Q}_{\mathsf{HI}}$, then returns $\perp$;
5. $b^* \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{AnCh}_b}}(st)$;
6. If $b = b^*$, return 1;
7. Return 0.

Fig. 5. Anonymity of DIHAC.

$Exp_b^{unl_{\mathsf{DIHAC}}}(\mathcal{A}, \lambda)$:
1. $(pp, msk, mpk) \leftarrow \mathsf{DIHAC.Setup}(1^\lambda)$;
2. $\mathcal{Q}_{\mathsf{HU}} \leftarrow \emptyset, \mathcal{Q}_{\mathsf{CU}} \leftarrow \emptyset, \mathcal{Q}_{\mathsf{HI}} \leftarrow \emptyset, \mathcal{Q}_{\mathsf{CI}} \leftarrow \emptyset, \mathcal{Q}_{\mathsf{Shw}} \leftarrow \emptyset, \mathcal{Q}_{\mathsf{RI}} \leftarrow \emptyset$;
3. $(i_0, u_0, i_1, u_1, \mathcal{D}^*, st) \leftarrow \mathcal{A}^{\mathcal{O}}(pp, mpk)$;
4. If $\exists j \in \mathcal{D}^*, a_j^{u_0} \neq a_j^{u_1}$, then returns $\perp$;
5. $b^* \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{UnCh}_b}}(st)$;
6. If $b = b^*$, return 1;
7. Return 0.

Fig. 6. Unlinkability of DIHAC.

credentials are unforgeable, if for any PPT adversary $\mathcal{A}$ having access to the oracles in $\mathcal{O} = \{\mathcal{O}_{\mathsf{Iss}_I}(i), \mathcal{O}_{\mathsf{ObtIss}_U}(u, \{a_j\}_{j\in[1,n]}), \mathcal{O}_{\mathsf{Cor}_U}(u), \mathcal{O}_{\mathsf{Iss}_U}(u, upk_u, \Pi_{2,u}, \{a_j\}_{j\in[1,n]}), \mathcal{O}_{\mathsf{Shw}}(u, \mathcal{D}, \mathsf{CTX})\}$, there is a negligible function $\epsilon(\lambda)$ such that:

$$Adv^{uf_{\mathsf{DIHAC}}} = \left|\Pr\left[Exp^{uf_{\mathsf{DIHAC}}}(\mathcal{A}, \lambda)\right] = 1\right| \leqslant \epsilon(\lambda)$$

*Anonymity:* Anonymity protects issuers from malicious users. In the anonymity experiment, the adversary can request credentials for all corrupted users by querying the oracle $\mathcal{O}_{\mathsf{Iss}_U}$ and corrupt any honest users by querying the oracle $\mathcal{O}_{\mathsf{Cor}_U}$. In the first stage of the experiment, the adversary outputs two issuer's indexes and one user's information. In the challenge stage, the adversary receives a credential from one of the two issuers for the user; as long as the attribute-based credential is valid, the adversary cannot determine with a non-negligible advantage which issuer issued the credential.

*Definition IV.2. Anonymity:* The anonymity is defined by experiment $Exp_b^{ano_{\mathsf{DIHAC}}}$ in Fig. 5. Issuer' public keys and credentials are anonymous, if for any PPT adversary $\mathcal{A}$ having access to the oracles in $\mathcal{O} = \{\mathcal{O}_{\mathsf{Iss}_I}(i), \mathcal{O}_{\mathsf{ObtIss}_U}(u, \{a_j\}_{j\in[1,n]}), \mathcal{O}_{\mathsf{Cor}_U}(u), \mathcal{O}_{\mathsf{Iss}_U}(u, upk_u, \Pi_{2,u}, \{a_j\}_{j\in[1,n]}), \mathcal{O}_{\mathsf{RelIss}}(u)\}$ and $\mathcal{O}_{\mathsf{AnCh}_b}(u, upk_u, \Pi_{2,u}, i_0, i_1, \{a_j\}_{j\in[1,n]})$, there is a negligible function $\epsilon(\lambda)$ such that:

$$Adv^{ano_{\mathsf{DIHAC}}} = \left|\Pr\left[Exp_1^{ano_{\mathsf{DIHAC}}}(\mathcal{A}, \lambda) = 1\right] - \frac{1}{2}\right| \leqslant \epsilon(\lambda)$$

*Unlinkability:* Unlinkability protects honest users from honest-but-curious verifiers. In the unlinkability experiment, the adversary can control all verifiers, corrupt issuers by querying the oracle $\mathcal{O}_{\mathsf{Cor}_I}$, issue credentials for all honest users by querying the oracle $\mathcal{O}_{\mathsf{Obt}_U}$, and obtain the issuer's identity of any user's credentials by querying the oracle $\mathcal{O}_{\mathsf{RelIss}}$. In the first stage of the experiment, the adversary outputs two user-issuer index pairs, and a selective disclosure policy. In the challenge stage, the adversary receives a presentation token from one of the two users; as long as the token is valid and consistent with the disclosure policy, the adversary cannot determine with a non-negligible advantage which user's credential is being used.

Note that the disclosed attributes of the two challenged users must be the same, otherwise the success of the adversary's challenge is trivial, since the adversary can directly determine the index of the user according to the disclosed attribute values.

*Definition IV.3. Unlinkability:* The unlinkability is defined by experiment $Exp_b^{unl_{\mathsf{DIHAC}}}$ in Fig. 6. Users' credentials are unlinkable, if for any PPT adversary $\mathcal{A}$ having access to the oracles in $\mathcal{O} = \{\mathcal{O}_{\mathsf{Iss}_I}(i), \mathcal{O}_{\mathsf{Cor}_I}(i), \mathcal{O}_{\mathsf{Obt}_U}(u, \{a_j\}_{j\in[1,n]}), \mathcal{O}_{\mathsf{Cor}_U}(u), \mathcal{O}_{\mathsf{Shw}}(u, \mathcal{D}, \mathsf{CTX})\}, \mathcal{O}_{\mathsf{RelIss}}(u)$ and $\mathcal{O}_{\mathsf{UnCh}_b}(\{u_0, i_0\}, \{u_1, i_1\}, \mathcal{D}, \mathsf{CTX})$, there is a negligible function $\epsilon(\lambda)$ such that:

$$Adv^{unl_{\mathsf{DIHAC}}} = \left|\Pr\left[Exp_1^{unl_{\mathsf{DIHAC}}}(\mathcal{A}, \lambda) = 1\right] - \frac{1}{2}\right| \leqslant \epsilon(\lambda)$$

### F. Generic Construction

In this section, we present a generic construction of DIHAC using our $\mathsf{TAM-Sign}$ scheme in combination with $\mathsf{SPS-EQ}$ scheme [46], and zero-knowledge signature of knowledge (ZKSoK) [23]. Our key ideas include: (1) The central authority of DIHAC computes an $\mathsf{SPS-EQ}$ signature on each issuer's public key and treats it as the issuer's credential. Since each issuer receiving an $\mathsf{SPS-EQ}$ signature can randomize their message (i.e., the issuer's public key) and corresponding signature (i.e., the issuer's credential), the issuer may issue credentials to users using the private key corresponding to the randomized public key, thereby hiding their identities from users. (2) Each user uses a user-generated witness as their private key, and a user-generated tag as their public key. To prevent replay attacks to credential verification, each user computes a fresh ZKSoK for their private key, and presents it to verifiers. (3) Upon receiving a user's tag and attributes, an issuer computes a $\mathsf{TAM-Sign}$ signature for each of the user's attributes on the user's tag under a randomized public key of the issuer and treats all $\mathsf{TAM-Sign}$ signatures as the user's credentials. For user credential verification, each user achieves user anonymity and issuer-hiding from any verifier by randomizing ① their issuer's credential/$\mathsf{SPS-EQ}$ signature and public key and ② their tag and $\mathsf{TAM-Sign}$ signatures (i.e., user credentials) corresponding to their disclosed attributes.

Below, we present a generic construction of DIHAC.

– $\mathsf{DIHAC.Setup}(1^\lambda) \rightarrow (pp, msk, mpk)$:

- Set the tag length $t \in \mathbb{Z}^+$ and the total number of user's attributes $n \in \mathbb{Z}^+$.
- Compute $pp_{\mathsf{TAM\text{-}Sign}} \leftarrow \mathsf{TAM\text{-}Sign.Setup}(1^\lambda, t, n)$.
- Using $pp_{\mathsf{TAM\text{-}Sign}}$ as parameters, compute $pp_{\mathsf{SPS\text{-}EQ}} \leftarrow \mathsf{SPS\text{-}EQ.Setup}(1^\lambda)$ to generate the remaining parameters of the $\mathsf{SPS-EQ}$ scheme.

*Remark IV.1:* This step implies a condition that the public key equivalence relation of $\mathsf{TAM-Sign}$ scheme $[pk]_{\mathcal{R}}$ is the same as that of the message equivalence relation of $\mathsf{SPS-EQ}$ scheme $[\vec{M}]_{\mathcal{R}}$, i.e. $[pk]_{\mathcal{R}} = [\vec{M}]_{\mathcal{R}}$.

- Using $pp_{\mathsf{TAM\text{-}Sign}}, pp_{\mathsf{SPS\text{-}EQ}}$ as parameters, compute $pp_{\mathsf{ZKSoK}} \leftarrow \mathsf{ZKSoK.Gen}(1^\lambda)$ to generate the remaining parameters of the ZKSoK scheme.
- Set $pp = (pp_{\mathsf{TAM\text{-}Sign}}, pp_{\mathsf{SPS\text{-}EQ}}, pp_{\mathsf{ZKSoK}})$.
- Compute $(msk, mpk) \leftarrow \mathsf{SPS\text{-}EQ.KeyGen}(pp, t \cdot n)$.

– $\mathsf{DIHAC.IKeyGen}(pp) \rightarrow (isk, ipk, \Pi_1)$:

- Compute $(isk, ipk) \leftarrow \mathsf{TAM\text{-}Sign.KeyGen}(pp)$.
- Compute $\Pi_1 \leftarrow \mathsf{ZKSoK.Sign}(isk, ipk)$.

– $\mathsf{DIHAC.Issue}_I(msk, mpk, ipk, \Pi_1) \rightarrow icred/\perp$:

- If $\mathsf{ZKSoK.Verify}(ipk, \Pi_1) = 0$, then output $\perp$.
- Compute $icred \leftarrow \mathsf{SPS\text{-}EQ.Sign}(ipk, msk)$.

– $\mathsf{DIHAC.VfCred}_I(mpk, isk, ipk, icred) \rightarrow 0/1$:

- If $\mathsf{SPS\text{-}EQ.Verify}(ipk, icred, mpk) = 1$, then output 1, otherwise output 0.

– $\mathsf{DIHAC.UKeyGen}(pp) \rightarrow (usk, upk, \Pi_2)$:

- Compute $(usk, upk) \leftarrow \mathsf{TAM.GenTag}(pp)$.
- Compute $\Pi_2 \leftarrow \mathsf{ZKSoK.Sign}(usk, upk)$.

– $\mathsf{DIHAC.Issue}_U(isk, ipk, icred, upk, \{a_j\}_{j\in[1,n]}, \Pi_2) \rightarrow (ipk', icred', ucred)/\perp$:

- If $\mathsf{ZKSoK.Verify}(upk, \Pi_2) = 0$, then output $\perp$.
- Choose a key converter $\mu_1 \in \mathbb{Z}_p^*$, and compute $(ipk', icred') \leftarrow \mathsf{SPS\text{-}EQ.ChgRep}(ipk, icred, \mu_1, mpk)$.

*Remark IV.2:* This step implies a conversion $ipk'$ algorithm, i.e $ipk' \leftarrow \mathsf{TAM\text{-}Sign.ConvertPK}(ipk, \mu_1)$.

- Compute $isk' \leftarrow \mathsf{TAM\text{-}Sign.ConvertSK}(isk, \mu_1)$.
- For all $j \in [1, n]$, compute $\sigma_j \leftarrow \mathsf{TAM\text{-}Sign.Sign}(isk', upk, a_j)$.
- Set $ucred = \{\sigma_j\}_{j\in[1,n]}$.

– $\mathsf{DIHAC.VfCred}_U(mpk, ipk', icred', usk, upk, \{a_j\}_{j\in[1,n]}, ucred) \rightarrow 0/1$:

- Check whether $\mathsf{SPS\text{-}EQ.Verify}(ipk', icred', mpk) = 1$.
- For all $j \in [1, n]$, check whether $\mathsf{TAM\text{-}Sign.Verify}(ipk', upk, \sigma_j, a_j) = 1$.
- Output 1 if the above checks hold and 0 otherwise.

– $\mathsf{DIHAC.Show}(mpk, ipk', icred', usk, upk, ucred, \{a_j\}_{j\in\mathcal{D}}, \mathsf{CTX}) \rightarrow tk$:

- Aggregate the signatures of disclosed attributes: $\sigma \leftarrow \mathsf{TAM\text{-}Sign.Aggr}(ipk', \{a_j, \sigma_j\}_{j\in\mathcal{D}}, upk)$.
- Choose a key converter $\mu_2 \in \mathbb{Z}_p^*$ and a tag converter $\rho \in \mathbb{Z}_p^*$.
- Update the issuer's public key and credential: $(ipk'', icred'') \leftarrow \mathsf{SPS\text{-}EQ.ChgRep}(ipk', icred', \mu_2, mpk)$.

*Remark IV.3:* This step implies a conversion $ipk''$ algorithm, i.e $ipk'' \leftarrow \mathsf{TAM\text{-}Sign.ConvertPK}(ipk', \mu_2)$.

- Convert the signature: $\sigma' \leftarrow \mathsf{TAM\text{-}Sign.ConvertSign}(\sigma, \mu_2)$.
- Change the tag representative: $(upk', \sigma'') \leftarrow \mathsf{TAM\text{-}Sign.ChgRep}(\sigma', upk, \rho)$.
- Compute a signature of knowledge: $\Pi_3 = \mathsf{ZKSoK.Sign}(usk, upk', \mathsf{CTX})$.
- Set $tk = (ipk'', icred'', upk', \sigma'', \Pi_3)$.

– $\mathsf{DIHAC.Verify}(mpk, \{a_j\}_{j\in\mathcal{D}}, tk, \mathsf{CTX}) \rightarrow 0/1$:

- Verify the presentation policy: If $\{a_j\}_{j\in\mathcal{D}}$ cannot satisfy the policy, then output 0.
- Verify $\Pi_3$: If $\mathsf{ZKSoK.Verify}(upk', \Pi_3, \mathsf{CTX}) = 0$, then output 0.
- Verify $(ipk'', icred'')$: If $\mathsf{SPS\text{-}EQ.Verify}(ipk'', icred'', mpk) = 0$, then output 0.
- Verify $(upk', \sigma'')$: If $\mathsf{TAM\text{-}Sign.Verify}(ipk'', upk', \sigma'', \{a_j\}_{j\in\mathcal{D}}) = 1$, then output 1, otherwise output 0.

*Remark IV.4:* For any given $\mathsf{TAM-Sign}$ scheme, $\mathsf{SPS-EQ}$ scheme, and ZKSoK scheme, our generic construction cannot be formed naturally unless the following two conditions are satisfied:

(1) As noted in Remark 4.1, the message equivalence relation for $\mathsf{SPS-EQ}$ must be the same as the public key equivalence relation for $\mathsf{TAM-Sign}$. This is because the CA uses $\mathsf{SPS-EQ}$ to compute the signature of an issuer's public key (i.e., issuer's credential), where the issuer's public key is input as the message in $\mathsf{SPS-EQ}$ scheme. Then, the issuer uses $\mathsf{TAM-Sign}$ to compute the signature of a user's attribute-tag pair (i.e., user's credential), where the issuer's public key is input as the public key in $\mathsf{TAM-Sign}$ scheme. Since the public key of an issuer is both the message input of $\mathsf{SPS-EQ}$ and the public key input of $\mathsf{TAM-Sign}$, the message equivalence relation of $\mathsf{SPS-EQ}$ must be identical to the public key equivalence relation of $\mathsf{TAM-Sign}$.

(2) As noted in Remarks 4.2 and 4.3, the $\mathsf{SPS-EQ.ChgRep}$ algorithm must imply the $\mathsf{TAM-Sign.ConvertPK}$ algorithm. First, updating the issuer's public key and updating the issuer's credential must occur using the same key converter; otherwise, the verification of $\mathsf{SPS-EQ.Verify}$ fails. Second, updating the issuer's public key and updating the user's credential must occur using the same key converter; otherwise, the verification of $\mathsf{TAM-Sign.Verify}$ fails. This means applying the same key converter to the issuer's public key, the issuer's credential, and the user's credential. Therefore, the execution of $\mathsf{SPS-EQ.ChgRep}$ algorithm includes the execution of $\mathsf{TAM-Sign.ConvertPK}$ algorithm.

### G. Security Analysis

In this section, we prove the security of our generic construction of DIHAC.

*Theorem IV.1:* Our DIHAC scheme is correct.

Naturally, if the $\mathsf{TAM-Sign}$ scheme, the $\mathsf{SPS-EQ}$ scheme and the ZKSoK scheme satisfy correctness, then our DIHAC scheme is correct.

*Theorem IV.2:* Our DIHAC scheme is unforgeable if the underlying $\mathsf{TAM-Sign}$ scheme is EUF-CMA secure and satisfies

unforgeability of tags, the SPS − EQ scheme is EUF-CMA secure, and the ZKSoK scheme is $SimExt\text{-}secure$.

*Proof:* There are three potential ways for an PPT adversary to win the unforgability game defined in Fig. 4: (1) it independently generates an issuer's key $(isk^*, ipk^*)$ and forgers a credential $icred^*$ issued by the CA for this issuer; or, (2) it independently generates a user's key $(usk^*, upk^*)$ and forges a credential $ucred^*$ issued by an authenticated issuer $i^*$ for this user; or, (3) it forges a tag (public key) $upk^*$ of an honest user $u^*$. The detailed proof is given in Supplemental Material C.2.1, available online.

*Theorem IV.3:* Our DIHAC scheme is anonymous if the underlying SPS − EQ scheme provides perfect adaption.

*Proof:* This straightforwardly follows from the signature adaptation of the SPS − EQ scheme.

*Theorem IV.4:* Our DIHAC scheme is unlinkable if the underlying TAM − Sign scheme and SPS − EQ scheme provide perfect adaption, the ZKSoK scheme is $SimExt\text{-}secure$, the SPS − EQ scheme has a class-hiding message space $\mathcal{M}$ (i.e., the public key space of TAM − Sign), and the TAM − Sign scheme has a class-hiding tag space $\mathcal{T}$.

*Proof:* Any PPT adversary cannot win the class-hiding experiments (Section II-C) of $\mathcal{M}$ and $\mathcal{T}$ with non-negligible probability. If the used TAM − Sign scheme and SPS − EQ scheme provide perfect adaption, and the ZKSoK scheme is $SimExt\text{-}secure$, the adversary who breaks the unlinkability of DIHAC scheme can be converted into a simulator against the class-hiding experiment of $\mathcal{M}$ and $\mathcal{T}$. The detailed proof is given in Supplemental Material C.2.2, available online.

### H. Concrete Instantiation and Extension

A possible instantiation of the DIHAC scheme can be obtained based on the TAM − Sign scheme in section III-C, the SPS − EQ scheme of Fuchsbauer et al. [46], and the Schnoor proof [19]. Our choices of TAM − Sign and SPS − EQ satisfy the conditions of Remark 4.4.

We can achieve traceability by letting the user with an identity $id$ generate a tracing key $utk = \widetilde{g}^{usk}$ for the issuer, who later can open the user's identity $id$ by checking whether $e(T_1', utk) = e(T_2', \widetilde{g})$.

With these choices, the DIHAC scheme is specified as follows.

– $\mathsf{DIHAC.Setup}(1^\lambda) \to (pp, msk, mpk)$:

- Set the tag length $t = 2$ and the total number of user's attributes $n \in \mathbb{Z}^+$.
- Compute $pp' \leftarrow \mathsf{TAM\text{-}Sign.Setup}(1^\lambda, 2, n)$:
  1) Generate a bilinear group $\mathcal{BL} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, p, g, \widetilde{g})$.
  2) Choose a pseudorandom function PRF and a collision resistant hash function $\mathsf{HASH}\{0,1\}^* \to \mathbb{G}_1^*$.
  3) Set the equivalence classes $[\vec{T}]_{\mathcal{R}} \subset (\mathbb{G}_1^*)^2$ and $[pk]_{\mathcal{R}} \subset (\mathbb{G}_2^*)^{2\cdot n}$.
  4) Set $pp' = (t{=}2, n, \mathcal{BL}, \mathsf{PRF}, \mathsf{HASH}, [\vec{T}]_{\mathcal{R}}, [pk]_{\mathcal{R}})$.
- Choose a collision resistant hash function $\mathsf{HASH}'$: $\{0,1\}^* \to \mathbb{Z}_p^*$, set $pp = \{pp', \mathsf{HASH}'\}$.
- Compute $(msk, mpk) \leftarrow \mathsf{SPS\text{-}EQ.KeyGen}(pp, 2 \cdot n)$:
  1) Choose $(y_{1,1}, y_{1,2}, \ldots, y_{n,1}, y_{n,2}) \xleftarrow{R} \mathbb{Z}_p^*$, for all $i \in [1,2], j \in [1,n]$ compute $Y_{j,i} = g^{y_{j,i}}$.
  2) Set $msk = (y_{1,1}, y_{1,2}, \ldots, y_{n,1}, y_{n,2})$, $mpk = (Y_{1,1}, Y_{1,2}, \ldots, Y_{n,1}, Y_{n,2})$.

– $\mathsf{DIHAC.IKeyGen}(pp) \to (isk, ipk, \Pi_1)$:

- Compute $(isk, ipk) \leftarrow \mathsf{TAM\text{-}Sign.KeyGen}(pp)$:
  1) Compute $rk \leftarrow \mathsf{PRF.KeyGen}(1^\lambda)$.
  2) For all $j \in [1,n]$, choose $(x_{j,1}, x_{j,2}) \xleftarrow{R} \mathbb{Z}_p^*$ and set $sk_j = (x_{j,1}, x_{j,2})$.
  3) Compute $pk_j = (\widetilde{X}_{j,1}, \widetilde{X}_{j,2}) \leftarrow (\widetilde{g}^{x_{j,1}}, \widetilde{g}^{x_{j,2}})$.
  4) Set $isk = (rk, sk_1, \ldots, sk_n)$ and $ipk = (pk_1, \ldots, pk_n)$.
- Compute a Schnorr proof $\Pi_1 = \mathsf{ZKSoK}\{(x_{1,1}, x_{1,2}, \cdots, x_{n,1}, x_{n,2}) \,|\, \forall i \in [1,2], j \in [1,n] : \widetilde{X}_{j,i} = \widetilde{g}^{x_{j,i}}\}$:
  1) Choose $(k_{1,1}, k_{1,2}, \ldots, k_{n,1}, k_{n,2}) \xleftarrow{R} \mathbb{Z}_p^*$, $\forall i \in [1,2]$, $j \in [1,n]$, and compute $\widetilde{R}_{j,i} = \widetilde{g}^{k_{j,i}}$.
  2) Compute $c = \mathsf{HASH}'(\widetilde{X}_{1,1}, \widetilde{X}_{1,2}, \ldots, \widetilde{X}_{n,1}, \widetilde{X}_{n,2}, \widetilde{R}_{1,1}, \widetilde{R}_{1,2}, \ldots, \widetilde{R}_{n,1}, \widetilde{R}_{n,2})$.
  3) $\forall i \in [1,2], j \in [1,n]$, compute $s_{j,i} = k_{j,i} - c \cdot x_{j,i}$.
  4) Set $\Pi_1 = \{c, s_{1,1}, s_{1,2}, \ldots, s_{n,1}, s_{n,2}\}$.

– $\mathsf{DIHAC.Issue}_I(msk, mpk, ipk, \Pi_1) \to icred/\perp$:

- If $\Pi_1$ verification fails, then output $\perp$:
  1) If $c' = \mathsf{HASH}'(\widetilde{X}_{1,1}, \widetilde{X}_{1,2}, \ldots, \widetilde{X}_{n,1}, \widetilde{X}_{n,2}, \widetilde{g}^{s_{1,1}} \widetilde{X}_{1,1}^c, \widetilde{g}^{s_{1,2}} \widetilde{X}_{2,2}^c, \ldots, \widetilde{g}^{s_{n,1}} \widetilde{X}_{n,1}^c, \widetilde{g}^{s_{n,2}} \widetilde{X}_{n,2}^c)$ the verification succeeds; otherwise, the verification fails.
- Compute $icred \leftarrow \mathsf{SPS\text{-}EQ.Sign}(ipk, msk)$:
  1) If $\widetilde{X}_{j,i} \notin \mathbb{G}_2^*$, for some $i \in [1,2], j \in [1,n]$, output $\perp$.
  2) Choose $y \xleftarrow{R} \mathbb{Z}_p^*$, compute $icred = (\widetilde{A}, B, \widetilde{B}) \leftarrow ((\prod_{j=1}^n \prod_{i=1}^2 \widetilde{X}_{j,i}^{y_{j,i}})^y, g^{1/y}, \widetilde{g}^{1/y})$.

– $\mathsf{DIHAC.VfCred}_I(mpk, isk, ipk, icred) \to 0/1$:

- Run $0/1 \leftarrow \mathsf{SPS\text{-}EQ.Verify}(ipk, icred, mpk)$:
  1) Check whether $\prod_{j=1}^n \prod_{i=1}^2 e(Y_{j,i}, \widetilde{X}_{j,i}) = e(B, \widetilde{A})$.
  2) Check whether $e(B, \widetilde{g}) = e(g, \widetilde{B})$.
  3) Output 1 if the above checks hold and 0 otherwise.

– $\mathsf{DIHAC.UKeyGen}(pp, id) \to (usk, upk, utk, \Pi_2)$:

- The user with identity $id$, sets $T_1 = \mathsf{HASH}(id) \in \mathbb{G}_1^*$.

*Remark IV.5:* $T_1$ is a natural random element of $\mathbb{G}_1^*$ associated with the user's identity $id$.

- Choose $usk \xleftarrow{R} \mathbb{Z}_p^*$, compute $T_2 = T_1^{usk}$ and $utk = \widetilde{g}^{usk}$, set $upk = (T_1, T_2)$.

*Remark IV.6:* $utk$ is the user's tracing key, which is sent to the issuer when the user requests attribute-based credentials.

- Compute a Schnorr proof $\Pi_2 = \mathsf{ZKSoK}\{usk \,|\, T_2 = T_1^{usk}, utk = \widetilde{g}^{usk}\}$:
  1) Choose $k \xleftarrow{R} \mathbb{Z}_p^*$, compute $R_1 = T_1^k, \widetilde{R}_2 = \widetilde{g}^k$.
  2) Compute $c = \mathsf{HASH}'(T_1, T_2, utk, R_1, \widetilde{R}_2)$.
  3) Compute $s = k - c \cdot usk$.
  4) Set $\Pi_2 = (c, s)$.

– $\mathsf{DIHAC.Issue}_U(isk, ipk, icred, icred, upk, id, utk, \{a_j\}_{j\in[1,n]}, \Pi_2) \to (ipk', icred', ucred)/\perp$:

- If $\Pi_2$ verification fails, then output $\perp$:
  1) If $c' = \mathsf{HASH}'(T_1, T_2, utk, g^s T_1^c, \widetilde{g}^s (utk)^c)$ the verification succeeds; otherwise, the verification fails.

- If $T_1 \neq \mathsf{HASH}'(id)$, then output $\perp$.
- Choose $\mu_1 \xleftarrow{R} \mathbb{Z}_p^*$, and compute $(ipk', icred') \leftarrow \mathsf{SPS\text{-}EQ.ChgRep}(ipk, icred, \mu_1, mpk)$:
  1) Compute $ipk' = ipk^{\mu_1}$.

*Remark IV.7:* This step is equivalent to executing the algorithm: $ipk' \leftarrow \mathsf{TAM\text{-}Sign.\ ConvertPK}(ipk, \mu_1)$.

1) Choose $\phi_1 \xleftarrow{R} \mathbb{Z}_p^*$, compute $icred' = (\widetilde{A}', B', \widetilde{B}') \leftarrow (\widetilde{A}^{\phi_1 \cdot \mu_1}, B^{1/\phi_1}, \widetilde{B}^{1/\phi_1})$.

*Remark IV.8:* The purpose of this step is to hide the identity of the issuer from the user by randomizing the issuer's public key $ipk$ and credentials $icred$.

- Compute $isk' \leftarrow \mathsf{TAM\text{-}Sign.ConvertSK}(isk, \mu_1)$.
  1) Compute $isk' = (rk, sk_1 \cdot \mu_1, \ldots, sk_n \cdot \mu_1)$, where $sk_j \cdot \mu_1 = (x_{j,1} \cdot \mu_1, x_{j,2} \cdot \mu_1)$.
- For all $j \in [1, n]$, compute $\sigma_j \leftarrow \mathsf{TAM\text{-}Sign.Sign}(isk', upk, a_j)$:
  1) Check $a_j \in \{0,1\}^*$ and $\vec{T} \in (\mathbb{G}_1^*)^2$.
  2) Generate a random $\gamma = \mathsf{PRF.Eval}(rk, \vec{T})$ and $\gamma \in \mathbb{Z}_p^*$.
  3) Compute $\sigma_j = (Z_j, Y_j, \widetilde{Y}_j, V_j) \leftarrow ((\prod_{i=1}^2 T_i^{x_{j,i} \cdot \mu_1})^\gamma, g^{1/\gamma}, \widetilde{g}^{1/\gamma}, \mathsf{HASH}(a_j)^{1/\gamma})$.
- Set $ucred = \{\sigma_j\}_{j \in [n]}$.
- Store $(id, utk)$ into the registration list $\mathcal{L}$.

*Remark IV.9:* $\mathcal{L}$ (initially $\emptyset$) is a list that stores users' registration information and is used by the issuer to trace users' identities.

– $\mathsf{DIHAC.VfCred}_U(mpk, ipk', icred', usk, upk, \{a_j\}_{j \in [1,n]}, ucred) \rightarrow 0/1$:

- Check whether $\mathsf{SPS\text{-}EQ.Verify}(ipk', icred', mpk) = 1$:
  1) Check whether $\prod_{j=1}^n \prod_{i=1}^2 e(Y'_{j,i}, \widetilde{X}'_{j,i}) = e(B', \widetilde{A}')$.
  2) Check whether $e(B', \widetilde{g}) = e(g, \widetilde{B}')$.
- For all $j \in [1, n]$, check whether $\mathsf{TAM\text{-}Sign.Verify}(ipk', upk, \sigma_j, a_j) = 1$:
  1) Check whether $\prod_{i=1}^2 e(T_i, \widetilde{X}'_{j,i}) = e(Z, \widetilde{Y})$.
  2) Check whether $e(Y, \widetilde{g}) = e(g, \widetilde{Y})$.
  3) Check whether $e(\mathsf{HASH}(a_j), \widetilde{Y}) = e(V, \widetilde{g})$.
- Output 1 if the above checks hold and 0 otherwise.

– $\mathsf{DIHAC.Show}(mpk, ipk', icred', usk, upk, ucred, \{a_j\}_{j \in \mathcal{D}}, \mathsf{CTX}) \rightarrow tk$:

- Compute $\sigma \leftarrow \mathsf{TAM\text{-}Sign.Agg}(ipk', \{a_j, \sigma_j\}_{j \in \mathcal{D}}, upk)$:
  1) Compute $\sigma = (Z, Y, \widetilde{Y}, V) \leftarrow (\prod_{j \in \mathcal{D}} Z_j, \quad Y_{j^*}, \widetilde{Y}_{j^*}, \prod_{j \in \mathcal{D}} V_j)$, where $j^* \in \mathcal{D}$.
- Choose $(\mu_2, \rho) \xleftarrow{R} \mathbb{Z}_p^*$, and compute $(ipk'', icred'') \leftarrow \mathsf{SPS\text{-}EQ.ChgRep}(ipk', icred', \mu_2, mpk)$:
  1) Compute $ipk'' = ipk'^{\mu_2}$.

*Remark IV.10:* This step is equivalent to executing the algorithm: $ipk'' \leftarrow \mathsf{TAM\text{-}Sign.ConvertPK}(ipk', \mu_2)$.

1) Choose $\phi_2 \xleftarrow{R} \mathbb{Z}_p^*$, compute $icred'' = (\widetilde{A}'', B'', \widetilde{B}'') \leftarrow (\widetilde{A}'^{\phi_2 \cdot \mu_2}, B'^{1/\phi_2}, \widetilde{B}'^{1/\phi_2})$.

- Compute $\sigma' \leftarrow \mathsf{TAM\text{-}Sign.ConvertSign}\ (\sigma, \mu_2)$:
  1) Choose $\phi_3 \xleftarrow{R} \mathbb{Z}_p^*$ and compute $\sigma' = (Z', Y', \widetilde{Y}', V') \leftarrow (Z^{\phi_3 \cdot \mu_2}, Y^{1/\phi_3}, \widetilde{Y}^{1/\phi_3}, V^{1/\phi_3})$.
- Compute $(upk', \sigma'') \leftarrow \mathsf{TAM\text{-}Sign.ChgRep}(\sigma', upk, \rho)$:
  1) Compute $\vec{T}' = \vec{T}^\rho = (T'_1, T'_2) \leftarrow (T_1^\rho, T_2^\rho)$.
  2) Choose $\phi_4 \xleftarrow{R} \mathbb{Z}_p^*$ and compute $\sigma'' = (Z'', Y'', \widetilde{Y}'', V'') \leftarrow (Z'^{\phi_4 \cdot \rho}, Y'^{1/\phi_2}, \widetilde{Y}'^{1/\phi_4}, V'^{1/\phi_4})$.
- Compute a Schnorr proof $\Pi_3 = \mathsf{ZKSoK}\{usk \mid T'_2 = T'^{usk}_1\}(\mathsf{CTX})$:
  1) Choose $k \xleftarrow{R} \mathbb{Z}_p^*$, compute $R = T'^k_1$.
  2) Compute $c = \mathsf{HASH}'(T'_1, T'_2, R, \mathsf{CTX})$.
  3) Compute $s = k - c \cdot usk$.
  4) Set $\Pi_3 = (c, s)$.
- Set $tk = (ipk'', icred'', upk', \sigma'', \Pi_3)$.

– $\mathsf{DIHAC.Verify}(mpk, \{a_j\}_{j \in \mathcal{D}}, tk, \mathsf{CTX}) \rightarrow 0/1$:

- If $\Pi_3$ verification fails, then output $\perp$:
  1) If $c' = \mathsf{HASH}'(T'_1, T'_2, g^s (T'_1)^c, \mathsf{CTX})$ the verification succeeds; otherwise, the verification fails.
- If $\mathsf{SPS\text{-}EQ.Verify}(ipk'', icred'', mpk) = 0$, then output 0:
  1) Check whether $\prod_{j=1}^n \prod_{i=1}^2 e(Y_{j,i}, \widetilde{X}''_{j,i}) = e(B'', \widetilde{A}'')$.
  2) Check whether $e(B'', \widetilde{g}) = e(g, \widetilde{B}'')$.
  3) Output 1 if the above checks hold and 0 otherwise.
- If $\mathsf{TAM\text{-}Sign.Verify}(ipk'', upk', \sigma'', \{a_j\}_{j \in \mathcal{D}}) = 1$, then output 1, otherwise output 0:
  1) If $\mathcal{D} \nsubseteq [1, n]$, output 0.
  2) Check whether $\prod_{i=1}^2 e(T'_i, \prod_{j \in \mathcal{D}} \widetilde{X}''_{j,i}) = e(Z'', \widetilde{Y}'')$.
  3) Check whether $e(Y'', \widetilde{g}) = e(g, \widetilde{Y}'')$.
  4) Check whether $e(\prod_{j \in \mathcal{D}} \mathsf{HASH}(a_j), \widetilde{Y}'') = e(V'', \widetilde{g})$.
  5) Output 1 if the above checks hold and 0 otherwise.

– $\mathsf{DIHAC.Trace}(\mathcal{L}, tk) \rightarrow id$:

- For each user $id \in \mathcal{L}$, this algorithm retrieves $utk$ from $\mathcal{L}$ and tests whether $e(T'_1, utk) = e(T'_2, \widetilde{g})$ until it gets a match, in which case it outputs the corresponding identity $id$.

*Remark IV.11:* This algorithm is operated by an issuer that takes his registration list $\mathcal{L}$ and a token $tk$ as inputs, and outputs a user's identity $id$.

## V. Function and Performance Evaluation

### A. Function Comparison

Table I summarizes a detailed comparison between our DIHAC scheme and related works, including three classical attribute-based credential schemes [32], [35], [36], eight delegatable anonymous credential schemes [23], [24], [25], [26], [27], [29], [31], [33], two issuer-hiding attribute-based credential schemes [16], [20], and an issuer-hiding anonymous credential scheme [30]. The comparisons are conducted in terms of the attribute-based, autonomous attribute, ZKP, concrete instantiation, issuer-hiding from users, issuer-hiding from verifiers, and traceability. *Attribute-based* means that the scheme is attribute-based. *Autonomous Attribute* means that users can obtain credentials of autonomous attributes rather than inheriting an issuer's attribute values. *ZKP* means what type of zero-knowledge proof is used to construct the scheme. Notably, both GS-Proof [28], and CH-Proof [22] rely on computationally expensive bilinear pairing operations, resulting in a higher computational overhead than Schnorr-Proof [19], which requires no such operations. "Generic ZKP" means that no specific type of ZKP is indicated in the literature. *Concrete Instantiation* means that a scheme

TABLE I
FUNCTION COMPARISON

| Scheme | Attribute-based | Autonomous Attribute | ZKP | Concrete Instantiation | Issuer-Hiding from Users | Issuer-Hiding from Verifiers | Traceability |
|---|---|---|---|---|---|---|---|
| [35] | √ | – | Schnorr-Proof | √ | × | × | × |
| [36] | √ | – | Schnorr-Proof | √ | × | × | × |
| [32] | √ | – | Schnorr-Proof | √ | × | × | × |
| [23] | × | × | Generic ZKP | × | √ | √ | × |
| [24] | × | × | GS-Proof | × | √ | √ | × |
| [25] | × | × | GS-Proof | × | √ | √ | × |
| [26] | × | × | GS-Proof | × | √ | √ | × |
| [27] | × | × | GS-Proof | × | √ | √ | × |
| [33] | × | × | Schnorr-Proof | √ | √ | √ | × |
| [29] | √ | √ | Schnorr-Proof | √ | × | √ | × |
| [31] | √ | × | Schnorr-Proof | √ | √ | √ | × |
| [20] | √ | √ | CH-Proof | √ | × | √ | × |
| [16] | √ | √ | Schnorr-Proof | √ | × | √ | × |
| [30] | × | × | Schnorr-Proof | √ | × | √ | × |
| DIHAC | √ | √ | Schnorr-Proof | √ | √ | √ | √ |

√: supported feature; ×: unsupported feature; –: not applicable

TABLE II
COMPUTATION AND STORAGE COMPARISON WITH CLASSICAL ATTRIBUTE-BASED CREDENTIAL SCHEMES

| Scheme | Issuing | | | Showing | | |
|---|---|---|---|---|---|---|
| | User | Issuer | Credential Size | User | Issuer | Token Size |
| CL [35] | $2(n+1)t_{e_1}$ | $(3n+6)t_{e_1}$ | $(2n+3)\lvert\mathbb{G}_1\rvert$ | $(2n+4)t_{e_1}+(n+2)t_{e_T}$ $+(n+3)t_p$ | $(n+3)t_{e_T}+(5n+5)t_p$ | $(n+2)\lvert\mathbb{Z}_p\rvert$ $+(2n+3)\lvert\mathbb{G}_1\rvert$ |
| BBS+ [36] | $2(n+1)t_{e_1}$ | $(n+4)t_{e_1}$ | $2\lvert\mathbb{Z}_p\rvert+\lvert\mathbb{G}_1\rvert$ | $6t_{e_1}+(n+4)t_{e_T}$ $+(n+6)t_{e_p}$ | $7t_{e_1}+(n+5)t_{e_T}$ $+(n+6)t_{e_p}$ | $(n+8)\lvert\mathbb{Z}_p\rvert+2\lvert\mathbb{G}_1\rvert$ |
| PS [32] | $2(n+1)t_{e_1}$ | $(n+4)t_{e_1}$ | $2\lvert\mathbb{G}_1\rvert$ | $2t_{e_1}+(n+1)t_{e_T}$ $+(n+3)t_{e_p}$ | $(n+2)t_{e_T}+(n+3)t_{e_p}$ | $(n+2)\lvert\mathbb{Z}_p\rvert+2\lvert\mathbb{G}_1\rvert$ |
| Section 4.8 | $2t_{e_1}+2t_{e_2}$ | $(3n+4)t_{e_1}$ $+(2n+5)t_{e_2}$ | $(2n+1)\lvert\mathbb{G}_1\rvert$ $+\lvert\mathbb{G}_2\rvert$ | $10t_{e_1}+(2n+4)t_{e_2}$ | $2t_{e_1}+(5n+7)t_{e_p}$ | $2\lvert\mathbb{Z}_p\rvert+6\lvert\mathbb{G}_1\rvert$ $+(2n+3)\lvert\mathbb{G}_2\rvert$ |

$n$: total number of user attributes

is instantiated, which makes it directly applicable without additional translation. *Issuer-Hiding from Users* means that a scheme allows issuers to issue user credentials without revealing the issuer's identity. *Issuer-Hiding from Verifier* means that a verifier cannot determine the issuer's identity of a token when validating tokens. *Traceability* means that a scheme supports tracing users' identities. Table I indicates that among all schemes in our comparison, only DIHAC (Section IV-H) supports all desirable features, making it a preferred solution in practice.

### B. Performance Evaluation

*1) Theoretical Evaluation:* Table II shows a theoretical comparison of our DIHAC scheme (Section IV-H) with three classical attribute-based credential schemes [32], [35], [36] in terms of computation and storage overhead, where $|\mathbb{G}_1|, |\mathbb{G}_2|, |\mathbb{G}_T|, |\mathbb{Z}_p|$ are the sizes of the elements in the group $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, and $\mathbb{Z}_p$, respectively; $t_{e_1}, t_{e_2}, t_{e_T}, t_p$ are the time costs for the exponential in the group $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, and pairing computations, respectively.

Table II shows that anonymous credentials based on the CL signature [35] have linear size and computational complexity. The schemes proposed in [36] and [32] exploit constant-size and randomizability of BBS+ and PS signatures, respectively, resulting in credentials and tokens of constant size. In addition, since hidden attributes cannot be revealed, these schemes require user to prove knowledge of $O(n)$ secret scalars involved in a pairing product equation during the credential showing, resulting in significant computational overhead. In comparison, while our DIHAC scheme also uses constant-size $\mathsf{TAM-Sign}$ scheme, it incorporates SPS-EQ to hide both issuer's public key and credentials (Table I shows that none of the aforementioned schemes support this functionality), resulting in linear size and computational complexity. Furthermore, our scheme leverages the aggregation feature of $\mathsf{TAM-Sign}$ to compute the attribute disclosure proof, eliminating the need for computationally complex zero-knowledge proofs and enabling more efficient credential showing compared to other schemes.

In Tables III and IV, we show a theoretical comparison of our DIHAC scheme (section IV-H) with the optimal delegatable credential scheme [29] and the most efficient issuer-hiding scheme [16] in terms of computation and storage overhead.

As shown in Table III, for Setup algorithm, the computational complexity of our scheme is the highest, but this algorithm only needs to be executed once for the system. For IKeyGen, $\mathsf{Issue}_I$ and UKeyGen, the computational complexity of our scheme is higher than that of [29] and [16], but fortunately for a new issuer or user these algorithms only need to be executed once. For $\mathsf{VfCred}_I$, the computational overhead of our scheme are

TABLE III
COMPUTATION OVERHEAD COMPARISON WITH IHAC SCHEMES

| **Algorithms** | Section 4.8 | [29] | [16] |
|---|---|---|---|
| Setup | $2n \cdot t_{e_1}$ | $1t_{e_2}$ | – |
| IKeyGen | $4n \cdot t_{e_2}$ | $1t_{e_1}$ | $1t_{e_2}$ |
| $\mathsf{Issue}_I$ | $1t_{e_1} + (6n+1)t_{e_2}$ | $(3n+4)t_{e_1} + 1t_{e_2}$ | $1t_{e_1} + 4t_{e_2}$ |
| $\mathsf{VfCred}_I$ | $(2n+3)t_p$ | $(3+3n)t_p$ | $6t_p$ |
| UKeyGen | $2t_{e_1} + 2t_{e_2}$ | $1t_{e_2}$ | $1t_{e_1}$ |
| $\mathsf{Issue}_U$ | $(3n+4)t_{e_1} + (2n+5)t_{e_2}$ | $1t_{e_1} + (3n+4)t_{e_2}$ | $(4+n)t_{e_1} + 1t_{e_2}$ |
| $\mathsf{VfCred}_U$ | $(5n+7)t_p$ | $(3+3n)t_p$ | $n \cdot t_{e_1} + 6t_p$ |
| Show | $10t_{e_1} + (2n+4)t_{e_2}$ | $1t_{e_1} + (3n+6)t_{e_2} + (3n+5)t_{e_T}$ | $(n+9)t_{e_1} + 5t_{e_2} + 4t_{e_T} + 7t_p$ |
| Verify | $2t_{e_1} + (5n+7)t_p$ | $(n+5)t_{e_T} + (8n+12)t_p$ | $(n+7)t_{e_1} + 9t_{e_T} + 12t_p$ |

TABLE IV
STORAGE COMPLEXITIES COMPARISON WITH IHAC SCHEMES

| **Variates** | Section 4.8 | [29] | [16] |
|---|---|---|---|
| $msk$ | $2n\|\mathbb{Z}_p\|$ | $1\|\mathbb{Z}_p\|$ | – |
| $pp$ & $mpk$ | $(2n+1)\|\mathbb{G}_1\| + 1\|\mathbb{G}_2\|$ | $n\|\mathbb{G}_1\| + 1\|\mathbb{G}_2\|$ | $(n+2)\|\mathbb{G}_1\| + 2\|\mathbb{G}_2\|$ |
| $isk$ | $(n+1)\|\mathbb{Z}_p\|$ | $1\|\mathbb{Z}_p\|$ | $1\|\mathbb{Z}_p\|$ |
| $ipk$ | $2n\|\mathbb{G}_2\|$ | $1\|\mathbb{G}_1\| + n\|\mathbb{G}_2\|$ | $1\|\mathbb{G}_2\|$ |
| $icred$ | $\|\mathbb{G}_1\| + 2\|\mathbb{G}_2\|$ | $1\|\mathbb{G}_1\| + (n+1)\|\mathbb{G}_2\|$ | $1\|\mathbb{G}_1\| + 2\|\mathbb{G}_2\|$ |
| $usk$ | $1\|\mathbb{Z}_p\|$ | $1\|\mathbb{Z}_p\|$ | $1\|\mathbb{Z}_p\|$ |
| $upk$ | $2\|\mathbb{G}_1\|$ | $n\|\mathbb{G}_1\| + 1\|\mathbb{G}_2\|$ | $1\|\mathbb{G}_1\|$ |
| $ucred$ | $(2n+1)\|\mathbb{G}_1\| + \|\mathbb{G}_2\|$ | $(n+1)\|\mathbb{G}_1\| + 1\|\mathbb{G}_2\|$ | $2\|\mathbb{G}_1\| + 1\|\mathbb{G}_2\|$ |
| $tk$ | $2\|\mathbb{Z}_p\| + 6\|\mathbb{G}_1\| + (2n+3)\|\mathbb{G}_2\|$ | $2\|\mathbb{Z}_p\| + (2n+4)\|\mathbb{G}_1\| + (2n+3)\|\mathbb{G}_2\|$ | $(n+5)\|\mathbb{Z}_p\| + 3\|\mathbb{G}_1\| + 4\|\mathbb{G}_2\|$ |

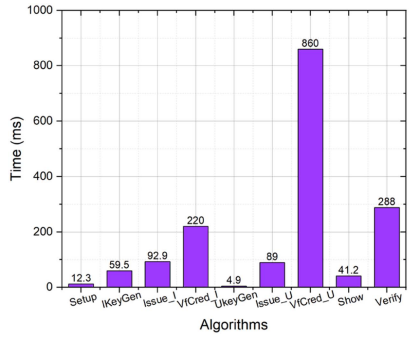
higher than that of [16], but lower than that of [29]. For $\mathsf{Issue}_U$, $\mathsf{VfCred}_U$, Show and Verify, our scheme achieves the same linear progressive complexity $O(n)$ as the schemes in [29] and [16].

As shown in Table IV, for $msk$, $pp\&$; $mpk$, $isk$, $ipk$ and $ucred$, the storage complexity of our scheme is higher than that of [29] and [16]. Fortunately, only one copy of these variables is required for each issuer or user. For $icred$ and $tk$, the storage overhead of our scheme is higher than that of [16], but lower than that of [29].

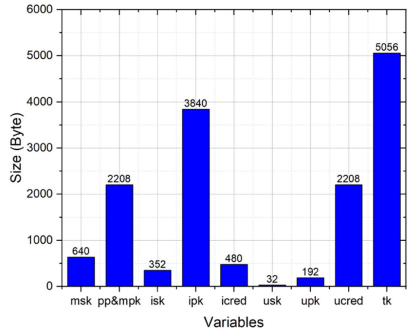
*2) Evaluation on a PC:* We implement the DIHAC instantiation using MIRACL [53], Type-III pairing and the Barreto-Naehrig curve (BN-256) [54] and test the system's performance at AES-100 b security level. The source code of our implementation is available at [55]. We run our implementation on a personal laptop (HUAWEI Matebook 14) with an AMD Ryzen-5 4600H with Radeon Graphics 3.00 GHz CPU, 16 GB RAM, 512 GB SSD running Ubuntu Kylin 16.04 operating system.

In Fig. 7 we show the computational and storage overheads of all algorithms at $n = 10$, where each timing result is computed as an average over 50 iterations. As shown in Fig. 7(a), in our DIHAC scheme, the Setup algorithm takes 12.3 ms. For key generation, IKeyGen and UKeyGen take 59.5 ms and 4.9 ms, respectively. To issue credentials for issuers, $\mathsf{Issue}_I$ and $\mathsf{VfCred}_I$ take 92.9 ms and 220 ms, respectively. To issue credentials to users, $\mathsf{Issue}_U$ and $\mathsf{VfCred}_U$ take 89 ms and 860 ms, respectively. For presenting tokens, Show and Verify take 41.2 ms and 288 ms, respectively. The most time-consuming of these is the $\mathsf{VfCred}_U$ algorithm, but it only needs to be executed once for a new user. In addition, the Show and Verify algorithms, which are executed most frequently, are efficient.

As shown in Fig. 7(b), in our DIHAC scheme, the storage of the public parameters $pp\&$; $mpk$ takes 2208 bytes. The



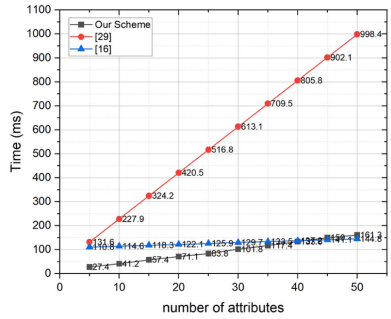

(a) Execution Time of Algorithms



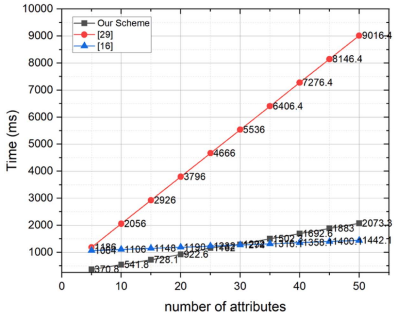

(b) Storage Size of Algorithms

Fig. 7. Execution time and storage size of algorithms.

storage overheads of the issuer's public key $ipk$ and credential $icred$ are 3840 bytes and 480 bytes, respectively. The storage overhead of the user's public key $upk$ and credential $ucred$ is 192 bytes and 2208 bytes, respectively. The storage of the presentation token $tk$ takes 5056 bytes. Among them, $tk$ takes




(a) Execution Time on PC




(b) Execution Time on Smartphone

Fig. 8. Execution time of show.

the most storage, which is 5056 bytes, but this is not a significant burden for communication and storage.

In Fig. 8(a), we compare the computational time of the most frequently used Show algorithm between our scheme and the schemes in [29] and [16], where the number of attributes varies from 5 to 50. Compared with the scheme in [29], our algorithm is at least 380% faster. We note that when the number of user attributes is around 30, our scheme is still faster than the scheme in [16], which uses seven bilinear pair operations, whereas our algorithm uses none.

*3) Evaluation on a Smartphone:* Since an anonymous credential system on the user side is usually deployed on a smartphone, we have measured its performance on a smartphone (HUAWEI Honor 9i) with a Hisilicon Kirin 659 (ARMv8-A) 2.36 GHz and 1.7 GHz CPU, 4 GB RAM running Andriod 9.0 operating system.

In Fig. 8(b), we compare the computational time of the Show algorithm on a smartphone between our scheme and the schemes in [29] and [16], where the number of attributes varies from 5 to 50. Compared with the scheme in [29], our algorithm is at least 200% faster. As long as the number of user attributes is less than 30, our scheme is faster than the scheme in [16].

The above analysis and comparison indicate that our scheme achieves stronger privacy protection than the classical attribute credential scheme [32], [35], [36]. Our DIHAC scheme has significantly lower computational overhead compared to the state-of-the-art delegatable credential scheme [29]; the efficiency of our scheme is also better than that of the state-of-the-art issuer-hiding scheme [16] as long as the number of attributes is no more than 30 in our experiments. In particular, our scheme has all the advantages of the scheme in [16] and additionally supports issuer-hiding from users; on the other hand, the execution time of Show in our scheme may grow to be longer than the scheme in [16] if the number of attributes is exceedingly large.

## VI. Related Work

### A. Issuer-Hiding Attribute-Based Credentials

Bobolz et al. [16] introduced the notion of issuer-hiding attribute-based credential (IHAC) to address the problem of hiding the identity of credential issuer from verifiers. In their scheme, a user presenting their credentials to a verifier hides the issuer of their credentials from the verifier by leveraging a set of access policies which includes multiple issuers' public keys signed by the verifier. They proposed a formal framework for the IHAC scheme, a generic construction and a concrete instantiation based on the structure-preserving signature scheme by Groth [17], Pedersen Hash [18], and the Schnorr-style proof of knowledge [19]. Their scheme achieves issuer-hiding from verifiers, but not in a strong sense because the anonymity of the issuer of any user's credentials in a successful verification by a verifier may diminish if the verifier creates an access policy set with a small number of issuers. Furthermore, their scheme does not support issuer-hiding from users. Conolly et al. [20] introduced the notion of signer-hiding, which is similar to the notion of Bobolz et al.'s issuer-hiding. They presented a signer-hiding attribute-based credential (SHAC) scheme from a new mercurial signature, which improved from the structure-preserving signatures on equivalence classes (SPS-EQ) because it can randomize not only any signed message and corresponding signature but also the public key used to verify the signature. Their scheme can be used to achieve issuer-hiding from verifiers without relying on the Generic Group Model (GGM) [21]. Their scheme extended the set-commitment scheme from [46] in combination with a new malleable NIZK argument from [22] to achieve selective disclosure proofs, it thus required a large number of bilinear pairing operations to be computed for presenting users' credentials. Their construction requires an OR-Proof to prove that an issuer's key is among a set of keys accepted by a verifier when a user's credentials issued by the issuer is verified by the verifier. This leads to a high computational cost for credential verification because it is linear in the number of issuers. In addition, their scheme does not support issuer-hiding from users.

Bosk et al. [30] proposed a different approach to formalize hidden issuer anonymous credentials (HIAC). Specifically, the authors introduced a new cryptographic primitive called $aggregator$, which can hide the issuer of a credential in a set of issuers. The proposed solution does not require any trusted settings. However, it is not attribute-based and lacks the ability for issuer-hiding from users.

### B. Delegatable Anonymous Credentials

Delegatable anonymous credentials (DACs) [23], [24], [25], [26], [27], [29], [31], [33] offer an alternative to address the problem of issuer-hiding from both users and verifiers, as some of them support privacy for issuers and users during credential delegation and presentation. In fact, such DACs with two entities

in their certification chains, in which non-root issuers obtain their credentials directly from a root-issuer, and users obtain their credentials from non-root issuers, can be regarded as double issuer-hiding anonymous credentials schemes.

Chase and Lysyanskaya [23] proposed the first DAC scheme based on generic zero-knowledge signature of knowledge, but their construction incurred a blow-up in that the size of a credential was exponential in the length of the credential's certification chain. Their scheme is not attribute-based, which is necessary for many practical applications.

Later, Belenkiy et al. [24], Fuchsbauer et al. [25], and Chase et al. [26], [27] published various constructions of DACs, but none is highly efficient for practical deployment for three main reasons. First, they used complex Groth-Sahai proofs [28] involving many expensive pairing operations. Second, their schemes are not attribute-based, which limits their applications due to not supporting fine-grained anonymous access control. Finally, they provided no concrete instantiations.

Camenisch et al. [29] proposed a delegatable attribute-based anonymous credentials (DAAC) derived from structure-preserving signature scheme [17], sibling signatures [29], and the Schnorr-style proof of knowledge [19]. Unlike our scheme, their scheme achieved efficiency and practicality by giving up the anonymity for delegators, i.e., providing no issuer-hiding from users. Later, Blomer et al. [31] published a DAAC scheme. They deviated from the usual approach of embedding a certificate chain in each credential and instead used a new primitive named dynamically malleable signatures, which is instantiated based on Pointcheval-Sanders signatures [32]. Their scheme allows users to remain anonymous while delegating and receiving a credential, thus providing issuer-hiding from both users and verifiers. However, unlike our scheme, their scheme does not allow users to apply for credentials corresponding to their autonomous attributes; instead, users can only apply for credentials corresponding to a subset of the delegator's attributes, which greatly limits its applications.

More recently, Crites et al. al [33] published a DAC scheme derived from mercurial signatures that can randomize not only a signed message and corresponding signature but also the public key. Their scheme is efficient; however, their scheme is not attribute-based.

### C. Selective Disclosure Credentials

Attribute-based anonymous credentials supporting selective disclosure have been designed with the following four approaches according to the properties of the underlying signature schemes. (1) The knowledge of hidden attributes is proved using zero-knowledge proofs (CL-signatures [34], [35], BBS-signatures [36], PS-signatures [32], [37], Groth-signatures [16]). (2) Users modify any signature on a set of attributes so that it is still verifiable even after removing a subset of disclosed attributes (unlinkable redactable signatures [38], [39], structure-preserving signatures on equivalence classes and set commitments [20], [45], [46]). (3) Users modify some attributes hidden in a signature to their default values (sanitizable signatures [40]). (4) Users receive one credential per attribute so that they can combine their credentials for the attributes to be disclosed (aggregate signatures [41], [42], [47]). Among them, the fourth approach has the highest computational efficiency because it does not need to prove the knowledge of hidden attributes or the validity of any modified signatures. The main computation cost of this approach is to aggregate credentials of disclosed attributes, which eliminates the need for users to provide complex zero knowledge proofs or compute bilinear pairing operations. Its disadvantage is that the size of each user's credentials is linear to the number of attributes, which is expensive for users to store their credentials on resource-constrained devices. We note that our scheme is an advancement in this approach.

### D. Structure-Preserving Signatures on Equivalence Classes and Mercurial Signatures

Hanser et al. [45] introduced a notation of structure-preserving signatures on equivalence classes (SPS-EQ), which can randomize both signed message and corresponding signature simultaneously. Given a prime order group $\mathbb{G}$ and a projective space $(\mathbb{G}^*)^l$, they defined projective equivalence classes of messages $[\vec{M}]_{\mathcal{R}}$ based on the equivalence relation: $\mathcal{R}_{\vec{M}} = \{(\vec{M}, \vec{M}') \in (\mathbb{G}^*)^l \times (\mathbb{G}^*)^l | \exists s \in \mathbb{Z}_p^* : \vec{M}' = \vec{M}^s\}$. They formalized the security of SPS-EQ, defined as $signature\ adaptation$, such that randomized signatures are distributed like fresh signatures on any new representative of equivalence classes. Subsequently, Fuchsbauer et al. [46] proposed a more streamlined SPS-EQ scheme in the generic group model (GGM) [21] and constructed a constant-size anonymous credentials based on it. The signature size of their scheme is only three group elements, and only two bilinear pairings are required for signature verification, which is the most efficient SPS-EQ scheme to date.

On the basis of Fuchsbauer et al.'s work, Hanzlik and Slamanig [47] introduced an approach of aggregation signature and proposed an aggregatable attribute-based equivalence class (AAEQ) signature. The AAEQ signature is aggregatable such that any signatures on multiple attributes for the same representative $\vec{M}$ of an equivalence class can be aggregated into a compact signature. They used AAEQ signature to construct a core/helper anonymous credentials (CHAC) to achieve a more efficient selective disclosure proof than the scheme in [46].

Crites and Lysyanskayaet [33], [43] further extended the idea of SPS-EQ and presented a new type of signature called mercury signature, which can be used to randomize not only any signed message and corresponding signature, but also the public key for verifying the signature. In this work, we use mercury signature [33] and AAEQ signature [47], in constructing $\mathsf{TAM-Sign}$ signature.

## VII. Conclusion and Future Work

In this paper, we introduced DIHAC to solve the problem of issuer-hiding from both users and verifiers in an attribute-based credential scheme. We gave a generic construction and an efficient instantiation using the $\mathsf{TAM-Sign}$ scheme we proposed and the $\mathsf{SPS-EQ}$ scheme [46]. We implemented DIHAC on a PC and smartphone to demonstrate its practicability.

The computational and storage complexity of DIHAC for a user presenting a credential is linear in the number of user attributes, which may limit its applications. We plan to address this issue in the future.

## REFERENCES


[1] D. Chaum, "Security without identification: Transaction systems to make big brother obsolete," *Commun. ACM*, vol. 28, no. 10, pp. 1030–1044, 1985.

[2] C. Garman, M. Green, and I. Miers, "Decentralized anonymous credentials," *Cryptol. ePrint Arch.*, vol. 2013, 2013, Art. no. 622.

[3] M. Chase, S. Meiklejohn, and G. Zaverucha, "Algebraic MACs and keyed-verification anonymous credentials," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 1205–1216.

[4] G. Couteau and M. Reichle, "Non-interactive keyed-verification anonymous credentials," in *Proc. IACR Int. Workshop Public Key Cryptogr.*, Cham, Switzerland: Springer, 2019, pp. 66–96.

[5] J. Blömer et al., "Updatable anonymous credentials and applications to incentive systems," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 1671–1685.

[6] J. Bobolz et al., "Privacy-preserving incentive systems with highly efficient point-collection," in *Proc. 15th ACM Asia Conf. Comput. Commun. Secur.*, 2020, pp. 319–333.

[7] M. Chase, T. Perrin, and G. Zaverucha, "The signal private group system and anonymous credentials supporting efficient verifiable encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2020, pp. 1445–1459.

[8] J. Han, L. Chen, S. Schneider, H. Treharne, and S. Wesemeyer, "Privacy-preserving electronic ticket scheme with attribute-based credentials," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 4, pp. 1836–1849, Jul./Aug. 2019.

[9] J. Han, L. Chen, S. Schneider, H. Treharne, S. Wesemeyer, and N. Wilson, "Anonymous single sign-on with proxy re-verification," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 223–236, 2019.

[10] E. Androulaki et al., "Privacy-preserving auditable token payments in a permissioned blockchain system," in *Proc. 2nd ACM Conf. Adv. Financial Technol.*, 2020, pp. 255–267.

[11] E. Brickell, J. Camenisch, and L. Chen, "Direct anonymous attestation," in *Proc. 11th ACM Conf. Comput. Commun. Secur.*, 2004, pp. 132–145.

[12] E. Brickell and J. Li, "Enhanced privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities," in *Proc. ACM Workshop Privacy Electron. Soc.*, 2007, pp. 21–30.

[13] D. Khovratovich and J. Law, "Sovrin: Digital identities in the blockchain era," *Github Commit Jasonalaw*, vol. 17, pp. 38–99, 2017.

[14] European Union, "General data protection regulation," *Official J. Eur. Union*, vol. 49, 2016, Art. no. L119. [Online]. Available: https://gdpr-info.eu

[15] California, "California consumer privacy act," 2018. [Online]. Available: https://oag.ca.gov/privacy/ccpa

[16] J. Bobolz et al., "Issuer-hiding attribute-based credentials," in *Proc. Int. Conf. Cryptol. Netw. Secur.*, Cham, Switzerland: Springer, 2021, pp. 158–178.

[17] J. Groth, "Efficient fully structure-preserving signatures for large messages," in *Proc. Int. Conf. Theory Application Cryptol. Inf. Secur.*, Springer, Berlin, Heidelberg, 2015, pp. 239–259.

[18] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Proc. Annu. Int. Cryptol. Conf.*, Berlin, Germany: Springer, 1991, pp. 129–140.

[19] C. P. Schnorr, "Efficient signature generation by smart cards," *J. Cryptol.*, vol. 4, no. 3, pp. 161–174, 1991.

[20] A. Connolly, P. Lafourcade, and O. P. Kempner, "Improved constructions of anonymous credentials from structure-preserving signatures on equivalence classes," in *Proc. IACR Int. Conf. Pract. Theory Public-Key Cryptogr.*, 2021, pp. 409–438.

[21] V. Shoup, "Lower bounds for discrete logarithms and related problems," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, Berlin, Germany: Springer, 1997, pp. 256–266.

[22] G. Couteau and D. Hartmann, "Shorter non-interactive zero-knowledge arguments and ZAPs for algebraic languages," in *Proc. Annu. Int. Cryptol. Conf.*, Cham, Switzerland: Springer, 2020, pp. 768–798.

[23] M. Chase and A. Lysyanskaya, "On signatures of knowledge," in *Proc. Annu. Int. Cryptol. Conf.*, Berlin, Germany: Springer, 2006, pp. 78–96.

[24] M. Belenkiy et al., "Randomizable proofs and delegatable anonymous credentials," in *Proc. Annu. Int. Cryptol. Conf.*, Berlin, Germany: Springer, 2009, pp. 108–125.

[25] G. Fuchsbauer, "Commuting signatures and verifiable encryption," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, Berlin, Germany: Springer, 2011, pp. 224–245.

[26] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn, "Malleable signatures: Complex unary transformations and delegatable anonymous credentials," *Cryptol. ePrint Arch.*, Rep. 2013/179, 2013. Accessed: Sep 15, 2023. [Online]. Available: http://eprint.iacr.org/

[27] M. Chase, M. Kohlweiss, and A. Lysyanskaya, "Malleable signatures: New definitions and delegatable anonymous credentials," in *Proc. IEEE 27th Comput. Secur. Foundations Symp.*, 2014, pp. 199–213.

[28] J. Groth and A. Sahai, "Efficient non-interactive proof systems for bilinear groups," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, Berlin, Germany: Springer, 2008, pp. 415–432.

[29] J. Camenisch, M. Drijvers, and M. Dubovitskaya, "Practical UC-secure delegatable credentials with attributes and their application to blockchain," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 683–699.

[30] D. Bosk et al., "Hidden issuer anonymous credential," in *Proc. Privacy Enhancing Technol.*, vol. 4, pp. 571–607, 2022.

[31] J. Blömer and J. Bobolz, "Delegatable attribute-based anonymous credentials from dynamically malleable signatures," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.*, Cham, Switzerland: Springer, 2018, pp. 221–239.

[32] D. Pointcheval and O. Sanders, "Short randomizable signatures," in *Proc. Cryptographers' Track RSA Conf.*, 2016, pp. 111–126.

[33] E. C. Crites and A. Lysyanskaya, "Delegatable anonymous credentials from mercurial signatures," in *Proc. Cryptographers' Track RSA Conf.*, Chamm, Switzerland: Springer, 2019, pp. 535–555.

[34] J. Camenisch and A. Lysyanskaya, "A signature scheme with efficient protocols," in *Proc. Int. Conf. Secur. Commun. Netw.*, Berlin, Germany: Springer, 2002, pp. 268–289.

[35] J. Camenisch and A. Lysyanskaya, "Signature schemes and anonymous credentials from bilinear maps," in *Proc. Annu. Int. Cryptol. Conf.*, Berlin, Germany: Springer, 2004, pp. 56–72.

[36] M. H. Au, W. Susilo, and Y. Mu, "Constant-size dynamic k-TAA," in *Proc. Int. Conf. Secur. Cryptogr. Netw.*, Berlin, Germany: Springer, 2006, pp. 111–125.

[37] A. Sonnino, M. Al-Bassam, S. Bano, S. Meiklejohn, and G. Danezis, "Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers," in *Proc. 26th Ann. Netw. Distrib. Syst. Secur. Symp., NDSS 2019*, San Diego, California, USA: The Internet Society, Feb. 24-27, 2019.

[38] J. Camenisch et al., "Composable and modular anonymous credentials: Definitions and practical constructions," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, Berlin, Germany: Springer, 2015, pp. 262–288.

[39] O. Sanders, "Efficient redactable signature and application to anonymous credentials," in *Proc. IACR Int. Conf. Public-Key Cryptogr.*, Cham, Switzerland: Springer, 2020, pp. 628–656.

[40] S. Canard and R. Lescuyer, "Protecting privacy by sanitizing personal data: A new approach to anonymous credentials," in *Proc. 8th ACM SIGSAC Symp. Inf. Comput. Commun. Secur.*, 2013, pp. 381–392.

[41] S. Canard and R. Lescuyer, "Anonymous credentials from (indexed) aggregate signatures," in *Proc. 7th ACM Workshop Digit. Identity Manage.*, 2011, pp. 53–62.

[42] C. Hébant and D. Pointcheval, "Traceable constant-size multi-authority credentials," in *Proc. Int. Conf. Secur. Cryptography Netw.*, Cham: Springer International Publishing, 2022, pp. 411–434.

[43] E. C. Crites and A. Lysyanskaya, "Mercurial signatures for variable-length messages," in *Proc. Privacy Enhancing Technol.*, vol. 4, 2021, pp. 441–463.

[44] S. D. Galbraith, K. G. Paterson, and N. P. Smart, "Pairings for cryptographers," *Discrete Appl. Math.*, 2008, vol. 156, no. 16, pp. 3113–3121.

[45] C. Hanser and D. Slamanig, "Structure-preserving signatures on equivalence classes and their application to anonymous credentials," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, Berlin, Germany: Springer, 2014, pp. 491–511.

[46] G. Fuchsbauer, C. Hanser, and D. Slamanig, "Structure-preserving signatures on equivalence classes and constant-size anonymous credentials," *J. Cryptol.*, vol. 32, no. 2, pp. 498–546, 2019.

[47] L. Hanzlik and D. Slamanig, "With a little help from my friends: Constructing practical anonymous credentials," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2021, pp. 2004–2023.

[48] P. Fauzi et al., "Quisquis: A new design for anonymous cryptocurrencies," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, Cham, Switzerland: Springer, 2019, pp. 649–678.
[49] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Proc. Conf. Theory Appl. Cryptographic Techn.*, Berlin, Germany: Springer, 1986, pp. 186–194.
[50] J. Camenisch and M. Stadler, "Efficient group signature schemes for large groups," in *Proc. Annu. Int. Cryptol. Conf.*, Springer, 1997, pp. 410–424.
[51] E. B. Sasson et al., "Zerocash: Decentralized anonymous payments from bitcoin," in *Proc. IEEE Symp. Secur. Privacy*, 2014, pp. 459–474.
[52] S. F. Sun et al., "Ringct 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero," in *Proc. Eur. Symp. Res. Comput. Secur.*, Cham, Switzerland: Springer, 2017, pp. 456–474.
[53] MIRACL Ltd., 2019. Accessed: Sep. 15, 2023. [Online]. Available: https://github.com/miracl/MIRACL
[54] J. Fan, F. Vercauteren, and I. Verbauwhede, "Faster $\mathbb{F}_p$-Arithmetic for cryptographic pairings on barreto-naehrig curves," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst.*, Berlin, Germany: Springer, 2009, pp. 240–253.
[55] DIHAC, 2023. Accessed: Sep. 15, 2023. [Online]. Available: https://github.com/DIHAC/DIHAC

**Rui Shi** received the MSc degree from Beijing Electronic Science and Technology Institute, Beijing, China, in 2015 and the PhD degree from the Beijing University of Posts and Telecommunications, Beijing, in 2023. He was a research engineer with Sinoinfosec Beijing Technology Company, Ltd., from 2015 to 2019. He is currently a research engineer with Beijing Electronic Science and Technology Institute. His research interests include the area of privacy protection and cryptography.

**Yang Yang** received the BSc degree from Xidian University, Xi'an, China, in 2006, and the PhD degree from Xidian University, China, in 2011. She is a senior research scientitst with the School of Computing and Information System, Singapore Management University, Singapore. She has been a full professor with the College of Computer Science and Big Data, Fuzhou University. Her research interests include information security and privacy protection. She has published more than 60 papers in *IEEE Transactions on Information Forensics and Security*, *IEEE Transactions on Dependable and Secure Computing*, *IEEE Transactions on Services Computing*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Industrial Informatics*, etc.

**Yingjiu Li** received the PhD degree from George Mason University, in 2003. He had been a faculty member with Singapore Management University, from 2003 to 2019. Now, he is ripple professor with the Department of Computer and Information Science, University of Oregon, Eugene, OR, USA. His research interests include IoT security and privacy, mobile security, and data security and privacy. He has published more than 170 papers in cybersecurity, and co-authored two academic books.

**Huamin Feng** is currently a professor with the Beijing Electronic Science and Technology Institute, Beijing University of Posts and Telecommunications, Beijing, China. His research interests include information security and cyberspace security.

**Guozhen Shi** is currently a professor with Beijing Electronic Science and Technology Institute, Beijing, China. His research interests include information security and cyberspace security.

**Hwee Hwa Pang** received the BSc (first class honors) and the MS degrees in computer science from the National University of Singapore, in 1989 and 1991, respectively, and the PhD degree in computer science from the University of Wisconsin-Madison, in 1994. He is a professor with the School of Computing and Information Systems, Singapore Management University, Singapore. His current research interests include database management systems, data security, and information retrieval.

**Robert H. Deng** (Fellow, IEEE) is an AXA chair professor in cybersecurity with the School of Computing and Information Systems, Singapore Management University, Singapore. His research interests include data security, network and system security. He has served/is serving on the editorial boards of many international journals in security, such as *IEEE Transactions on Information Forensics and Security*, *IEEE Transactions on Dependable and Secure Computing*, etc.