Efficient Aggregate Anonymous Credentials for Decentralized Identity

Rebekah Mercer[1,2]✉, Kaoutar Elkhiyaoui[1], Angelo De Caro[1], and Elli Androulaki[1]

[1] IBM Research - Zurich
{kao, adc, lli}@zurich.ibm.com
[2] ETH Zurich
rmercer@student.ethz.ch

Anonymous credential schemes allow users to prove claims about themselves in a privacy-preserving manner, enabling the verification of certified attributes (e.g., citizenship) without revealing additional information. Traditional schemes, including those supporting multiple issuers, typically assume that each issuer can certify all user attributes. In contrast, in practice identity providers generally issue credentials for only a limited subset of attributes.

Aggregate anonymous credential schemes address this setting by allowing users to obtain credentials from multiple issuers, aggregate them, and prove claims about their attributes in a single shot. In this paper, we introduce tag-based aggregate signatures, a primitive that enables aggregation of signatures issued under a common tag. Using this primitive, we present a black-box construction of aggregate anonymous credentials based on standard digital signatures and signatures of knowledge, and prove its security.

We provide an efficient instantiation using BLS signatures [7], AGHO structure-preserving signatures [1], and Sigma protocol-based signatures of knowledge. An experimental evaluation demonstrates that the resulting scheme is practical and well suited for decentralized identity applications.

1 Introduction

Anonymous credential schemes [18,12] allow users to obtain credentials on attributes they possess and later prove possession to verifiers without revealing their identity or leaking hidden attribute values. The interactive protocol for obtaining credentials is called credential issuance, and the protocol for proving possession of valid credentials on certain attributes is called a credential presentation. Anonymous credential schemes aim to provide unforgeability of credentials and unlinkability of credential presentations. Unlinkability ensures that a verifier cannot determine whether two presentations come from the same user or from different users holding the same attributes, even if the issuer and verifier are the same. Unforgeability ensures that only credentials a user has legitimately obtained will successfully verify.

Early anonymous credential schemes typically assume a single issuer certifying all attributes, creating a single powerful entity. Later schemes explored

distributing issuance across multiple authorities or delegating issuance to sub-issuers, but conceptually the issuer, though now decentralized, is still responsible for all attributes [10,25,19]. This reduces the trust assumptions compared with using one party as issuer, but does not reflect real-world scenarios, where distinct entities independently certify one or a small subset of attributes.

In decentralized identity systems, different authorities each typically certify a small number of attributes. Examples are age, residency, citizenship, possession of a driving license, and possession of professional qualifications. Existing credential systems require coordination between issuers, which becomes impractical if there are many issuers worldwide, different subsets of which will be relevant to different users.

In this paper, we address this limitation by introducing an aggregate anonymous credential scheme that allows credentials issued by different issuers to be combined into a single, compact credential, enabling efficient proofs over user attribute values.

We consider a model in which each attribute is independently certified by a designated issuer, reflecting real-world scenarios in which distinct authorities (such as government agencies) are responsible for issuing specific credentials (e.g., driver's licenses or residency permits), and a separate registration authority manages user registration. A user starts their journey in the system by registering with a trusted registration authority, which binds the user's public key to a unique tag by producing a signature over the public key and the tag. This signature later serves as a proof of registration. When a registered user seeks to obtain an attribute credential from an issuer, she presents the registration signature and prove knowledge of the secret key corresponding to their registered public key. The issuer then signs the user's attribute value together with the user's tag using aggregate tag-based signatures. We emphasize that decentralization in our setting refers to issuance of attributes across independent issuers, not to the elimination of a registration authority.

The use of aggregate tag-based signatures ensures that only attribute credentials associated with the same tag can be aggregated. This prevents malicious users or issuers from combining credentials across identities or forging new credentials, while allowing honest users to aggregate credentials efficiently without further interaction with the issuers. To demonstrate possession of certified attribute values, the user aggregates the relevant attribute credentials and proves in zero-knowledge that (1) they know the secret key corresponding to a registered public key, (2) they hold a valid aggregate credential certifying the disclosed attribute values, and (3) the tag associated with the registered public key is the same as the tag associated with the aggregated credentials. At no point are the public key or the tag revealed, protecting user privacy.

Contributions. The contributions of this paper are as follows.

- We formally define aggregate tag-based signatures and propose an aggregate anonymous credential scheme that leverages, in a black-box manner, digital signatures, aggregate tag-based signatures and signatures of knowledge.

- We formalize the security of aggregate anonymous credentials in the real/ideal world paradigm, and prove the security of the resulting black-box construction assuming the constituent primitives are secure.
- We introduce an efficient instantiation of aggregate tag-based signatures and analyze its security in the random oracle model.
- We instantiate the black-box construction using AGHO structure-preserving signatures [1], our aggregate tag-based signature, and Schnorr-like signatures of knowledge, and evaluate its performance.

Layout. The paper is organized as follows. Section 2 states the problem, introduces the system participants, and formalizes the security requirements of aggregate anonymous credentials. Section 3 describes the relevant cryptographic building blocks, while Section 4 details a black-box construction of an aggregate anonymous credentials scheme. Section 5 describes an instantiation that leverages our tag-based aggregate signatures, and Section 6 evaluates its overall performances. Finally, Section 7 reviews the related work and Section 8 concludes the paper.

2 Aggregate Anonymous Credentials

2.1 System Participants and Threat Model

An aggregate anonymous credentials system involves participants described as follows.

First, the **registration authority (**RA**)** is equipped with a pair of secret and public keys (sk, pk), and tasked with registering and certifying the users' public keys. Notably, the registration authority is responsible for ensuring that each user's public key is unique and that the user holds the corresponding secret key. An example of a registration authority is the agency assigning social security numbers to the residents of a country.

Next, we have a set of **issuers** such that each issuer I has a pair of secret and public keys (isk, ipk) used to generate user credentials. For the sake of simplicity, we assume that each I produces a user credential for a single attribute $\mathtt{att}$, and that it is the only issuer for that attribute. In other words, each type of attribute is uniquely associated with an issuer. An example of an issuer is the agency in charge of issuing driving licenses.

We also have a set of **users** such that each user $\mathcal{U}$ is equipped with a pair of secret and public keys (usk, upk), and is required, from time to time, to *anonymously* convince a verifier $\mathcal{V}$ that the user holds values $\mathfrak{a}_1, ..., \mathfrak{a}_l$ of some set of attributes $\mathtt{att}_1, ..., \mathtt{att}_l$. To that end, user $\mathcal{U}$ will first register their public key by interacting with registration authority RA. Then, when the user would like to certify the value of a given attribute, they will contact the relevant issuer I. The latter returns a valid credential if the user meets the necessary requirements to hold the attribute value. The valid credentials that $\mathcal{U}$ obtains can be aggregated as needed so $\mathcal{U}$ can prove claims about their attribute values.

Finally, there are a set of **verifiers** $\{\mathcal{V}_1, \mathcal{V}_2, \ldots\}$ that check users' claims against the public key of the registration authority and the public keys of the issuers. A user claim, in this context, corresponds to a user $\mathcal{U}$ *provably* presenting the values of a subset of their attributes. We call the process of producing a user claim a *presentation*.

System Requirements. Similar to traditional anonymous credentials, an anonymous aggregate credentials system must guarantee the security properties of **unforgeability** and **unlinkability**. Unforgeability ensures that a user can produce a valid presentation of an attribute value only if they received a valid credential from the authorized issuer certifying said value. Unlinkability assures users that their presentations are *indistinguishable* between users holding the same attribute values. We note that we aim to achieve unlinkability between users' presentations to verifiers, and do not try to hide users' attribute values from the issuers themselves. Furthermore, anonymous aggregate credentials offer the added feature of **aggregatability**: a user receives a credential for each of their attribute values from the corresponding issuer, and at time of presentation, the user aggregates the necessary credentials and *efficiently* and *securely* disclose the relevant attribute values.

Threat Model. We assume that the **registration authority** is honest-but-curious: it follows the protocol correctly, ensures uniqueness of user registrations, and assigns a single tag per user, but may collude with other system participants to try to correlate presentations. For completeness, if the RA was malicious, it could intentionally assign duplicate tags or register users under adversarially chosen identities. Addressing a fully malicious RA is orthogonal to this work. **Issuers**, **users**, and **verifiers**, on the other hand, can be malicious: they may collude with other system participants and deviate *arbitrarily* from the specifications of the protocol to undermine the privacy of honest users, trick honest verifiers into accepting invalid presentations, or forge credentials. We note that honest verifiers only accept credentials from a predefined registered set of issuers, known by their public keys. This means that, by construction, a malicious user is not able to impersonate an issuer to an honest verifier.

2.2 Security Formalization

We formalize the security of anonymous aggregate credentials following the ideal/world paradigm, using a *standalone* simulation-based definition. In the *ideal world*, we find a functionality $\mathcal{F}$, which by construction, captures the desired security and privacy properties of anonymous aggregate credentials, together with a simulator $\mathcal{S}$ that controls a subset of issuers, users, and verifiers. On the other hand, in the *real world*, we find a candidate protocol Π and an adversary $\mathcal{A}$ that controls the same subset of system participants as $\mathcal{S}$, and can instruct them to behave *arbitrarily*. We use the real/ideal-world paradigm to capture all interactions between issuers, users, and verifiers in one instance.

In this paper, we consider a *static corruption model* that allows the simulator $\mathcal{S}$ (adversary $\mathcal{A}$ resp.) to corrupt any number of issuers, users and verifiers but before any call to $\mathcal{F}$'s interfaces (Π's functions resp.) takes place. We note that since the registration authority is honest-but-curious, $\mathcal{S}$ will not corrupt it, however, it will be given access to any information that the registration authority learns during its interactions with various system participants.

Ideal Functionality $\mathcal{F}$. The ideal functionality $\mathcal{F}$ represents a trusted third party, which executes the operations of aggregate anonymous credentials on behalf of system participants in a way that *perfectly* meets the system requirements. That is, $\mathcal{F}$ receives inputs from the system participants and simulator $\mathcal{S}$ and correctly generate the corresponding outputs, which are then transmitted to the parties authorized to access them. Accordingly, $\mathcal{F}$ exposes 5 interfaces, Register, Issue, Aggregate, Present and Verify, through which $\mathcal{S}$ and *honest* system participants submit their inputs.

At a high level, the ideal functionality captures the properties of aggregate anonymous credentials: unforgeability, unlinkability, and aggregatability. Unforgeability is enforced by ensuring that only validly issued attributes can be successfully presented or verified. Unlinkability is achieved as randomness is returned in each query to present, which prevents presentations from being correlated. Aggregation occurs through calling the Aggregate interface.

Register is called by a user $\mathcal{U}$ to register into the system. If registration authority RA agrees and $\mathcal{U}$ has never been registered before, then $\mathcal{U}$ is successfully registered.

Issue is invoked when a user $\mathcal{U}$ wants to get a credential that certifies value $\mathfrak{a}$ of some attribute $\mathtt{att}$. If the authorized issuer agrees to $\mathcal{U}$'s issuance request, then $\mathcal{U}$ obtains a credential that she can use in subsequent presentations. Without loss of generality, we assume that I_i produces credentials for attribute $\mathtt{att}_i$

During a Register call, $\mathcal{S}$ learns the identity of $\mathcal{U}$ whereas during an Issue call, $\mathcal{S}$ additionally learns the value of $\mathcal{U}$'s attribute.

Aggregate is invoked by a user $\mathcal{U}$ to aggregate the user's credentials for some attribute values $\boldsymbol{\mathfrak{a}}$ in a way that enables them to successfully present the aggregate credential later. Aggregation is a local operation, and hence, $\mathcal{S}$ does not learn any information during an Aggregate call. A successful Aggregate call results in $\mathcal{U}$ receiving a random string σ representing the aggregate credentials.

Present is queried by a user $\mathcal{U}$ to produce a presentation of some attribute values (denoted $\boldsymbol{\mathfrak{a}}$). $\mathcal{U}$'s request consists of triple $(\boldsymbol{\mathfrak{a}}, \sigma, \mu)$, whereby σ represents the aggregation of the credentials certifying $\boldsymbol{\mathfrak{a}}$ and μ is a nonce that identifies this presentation request. If $\mathcal{U}$ has previously succeeded in calling Aggregate with $\boldsymbol{\mathfrak{a}}$, then $\mathcal{F}$ accepts the query; otherwise, it checks if $\mathcal{U}$ actually holds the $\boldsymbol{\mathfrak{a}}$. This captures the fact that even if a corrupt $\mathcal{U}$ does not invoke Aggregate prior to calling Present, $\mathcal{F}$ will still accept the presentation request, as long as $\mathcal{U}$ has previously received the relevant credentials. If $\mathcal{F}$ accepts, then it generates a random string ϕ; the randomness of ϕ ensures that triple $(\boldsymbol{\mathfrak{a}}, \mu, \phi)$ discloses no information about $\mathcal{U}$ beyond what's revealed by $\boldsymbol{\mathfrak{a}}$.

Verify is invoked by a verifier $\mathcal{V}$ to check the validity of credential presentation $(\mathfrak{a}, \mu, \phi)$. If the presentation has already been produced by a valid Present call, then $\mathcal{F}$ accepts and returns `true`. If it has already been flagged as invalid, then $\mathcal{F}$ rejects and returns `false`. Finally, if the presentation has never been received before, then $\mathcal{F}$ verifies whether the presentation could have been created by corrupt users and issuers. That is, $\mathcal{F}$ checks if there are corrupt users that hold all the disclosed attribute values, and if not, then $\mathcal{F}$ also checks if there are corrupt issuers that can produce valid credentials for the attribute values not held by corrupt users. In these checks succeed, then $\mathcal{F}$ refers to $\mathcal{S}$ and returns what $\mathcal{S}$ returns; otherwise, it records the presentation, flags it as invalid, and outputs `false`.

Figure 1 depicts the details of ideal functionality $\mathcal{F}$. We assume there are N issuers in the system, and accordingly, the system supports N attributes. When $\mathcal{F}$ is called through Aggregate, Present, or Verify, it receives vector $\mathfrak{a} = (\mathfrak{a}_1, \ldots, \mathfrak{a}_N)$ such that $\mathfrak{a}_i = \varnothing$ or $\mathfrak{a}_i \in \{0,1\}^*$. We remark that $\mathfrak{a}_i = \varnothing$ indicates that the value of attribute $\mathtt{att}_i$ will be hidden during credential presentation, whereas $\mathfrak{a}_i \neq \varnothing$ indicates that value is disclosed.

Let $\textsc{Exec}_{\Pi,\mathcal{A}}(1^\kappa)$ denote the outputs of the honest parties and $\mathcal{A}$, during an execution of Π in the real world and $\textsc{Exec}_{\mathcal{F},\mathcal{S}}(1^\kappa)$ denote the outputs of the honest parties and $\mathcal{S}$, interacting with $\mathcal{F}$ in the ideal world.

Definition 1. *A protocol Π* securely realizes *ideal functionality $\mathcal{F}$ in the standalone model if, for every probabilistic polynomial time (PPT) adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ satisfying $\textsc{Exec}_{\Pi,\mathcal{A}}(1^\kappa) \approx_c \textsc{Exec}_{\mathcal{F},\mathcal{S}}(1^\kappa)$, where κ is the security parameter, and $\approx_c$ denotes computational indistinguishability.*

3 Building Blocks

3.1 Digital Signatures (DS)

A digital signature scheme consists of the following algorithms:

- $\mathsf{DS.KeyGen}(1^\kappa) \to (sk, pk)$: on input security parameter 1^κ, output a pair of secret and public keys (sk, pk).
- $\mathsf{DS.Sign}(sk, m) \to \psi$: on input secret key sk and message $m \in \{0,1\}^*$, output signature ψ.
- $\mathsf{DS.Verify}(pk, m, \psi) \to \{0,1\}$: on input public key pk, message m, and signature ψ, return 0 or 1.

A signature scheme is *existentially unforgeable* if, without the secret key, it is computationally infeasible for any PPT adversary to produce a valid signature.

3.2 Signatures of Knowledge (SoK)

Signatures of knowledge (SoK) allow a prover to convince a verifier that she knows a witness $\mathbb{w}$ for a public statement $\mathbb{x}$ while binding the proof to a message μ. The following definitions are based on those found in [17].

Register. On input $\mathsf{Register}$ from user $\mathcal{U}$, send $(\mathsf{Register}, \mathcal{U})$ to RA and wait for its confirmation. Upon receiving the confirmation, do:

1. If $\mathsf{Cred}[\mathcal{U}] \not\rightarrow \mathsf{nil}$, then abort; else continue. (*a user can be registered only once*)
2. Send $(\mathsf{Register}, \mathcal{U})$ to $\mathcal{S}$ and wait for the go-ahead; on receiving the go-ahead continue.
3. Set $\mathsf{Cred}[\mathcal{U}] \leftarrow [\varnothing, \ldots, \varnothing]$ ($\mathsf{Cred}[\mathcal{U}]$ is an array of length N, and $\mathsf{Cred}[\mathcal{U}][i]$ is the i^{th} element).
4. Return $\mathsf{RegisterEND}$ to $\mathcal{U}$.

IssueCred. On input $(\mathsf{IssueCred}, \mathtt{att}_i, \mathfrak{a}_i)$ from $\mathcal{U}$, with $\mathfrak{a}_i$ being value of attribute $\mathtt{att}_i$, send $(\mathsf{IssueCred}, \mathcal{U}, \mathfrak{a}_i)$ to issuer I_i associated with $\mathtt{att}_i$ and wait for its confirmation. Upon confirmation, do:

1. If issuer I_i is honest, then:
   (a) If $\mathsf{Cred}[\mathcal{U}] \rightarrow \mathsf{nil}$, then abort; else continue. *(An honest issuer will only issue credentials to a registered user.)*
   (b) If $\mathsf{Cred}[\mathcal{U}][i] \neq \varnothing$, then abort; else continue. *(An honest issuer will issue at most one valid credential per user.)*
   (c) Send $(\mathsf{Issue}, \mathcal{U}, \mathtt{att}_i, \mathfrak{a}_i)$ to $\mathcal{S}$ and wait for the go-ahead; on receiving the go-ahead, continue.
2. Else:
   (a) Send $(\mathsf{IssueCred}, \mathcal{U}, \mathtt{att}_i, \mathfrak{a}_i)$ to $\mathcal{S}$, and waits for its response.
   (b) If $\mathcal{S}$ sends Accept, then continue; else abort.
3. Set $\mathsf{Cred}[\mathcal{U}][i] \leftarrow \mathsf{Cred}[\mathcal{U}][i] \cup \{\mathfrak{a}_i\}$.
4. Return $(\mathsf{IssueEND}, \mathtt{att}_i, \mathfrak{a}_i)$ to $\mathcal{U}$ and $(\mathsf{IssueEND}, \mathcal{U}, \mathfrak{a}_i)$ to I_i.

Aggregate. On input $(\mathsf{Aggregate}, \mathfrak{a})$ from $\mathcal{U}$, where $\mathfrak{a} = (\mathfrak{a}_1, ..., \mathfrak{a}_N)$ and $\mathfrak{a}_i \in \{\varnothing\} \cup \{0,1\}^*$, do:

1. If $\mathsf{Cred}[\mathcal{U}] = \mathsf{nil}$, then abort; else continue. *(Unregistered users cannot successfully aggregate credentials.)*
2. If $\forall i \in [1 \,..\, N] : \mathfrak{a}_i = \varnothing \vee \mathfrak{a}_i \in \mathsf{Cred}[\mathcal{U}][i]$, then:
   (a) Randomly generate σ.
   (b) Set $\mathsf{AggCred}[\sigma] \leftarrow \langle \mathcal{U}, \mathfrak{a} \rangle$.
   (c) Return $(\mathsf{AggregateEND}, \sigma)$ to $\mathcal{U}$. *(Users who hold non-empty attribute values successfully aggregate the corresponding credentials.)*
3. Else abort.

Present. On input $(\mathsf{Present}, \mathfrak{a}, \sigma, \mu)$ from $\mathcal{U}$, where $\mathfrak{a} = (\mathfrak{a}_1, ..., \mathfrak{a}_N)$ s.t. $\mathfrak{a}_i = \varnothing$ if $\mathfrak{a}_i$ is not disclosed, do:

1. If $\mathsf{AggCred}[\sigma] \rightarrow \langle \mathcal{U}, \mathfrak{a} \rangle$, then:
   (a) Randomly generate ϕ.
   (b) Set $\mathsf{Presented}[\mathfrak{a}, \mu, \phi] \leftarrow \mathsf{true}$.
   (c) Return $(\mathsf{PresentEND}, \phi)$ to $\mathcal{U}$ and $(\mathsf{PresentEND}, \mathfrak{a}, \mu, \phi)$ to $\mathcal{S}$. *(Users who successfully aggregated their credentials successfully sign messages.)*
2. Else if $\mathsf{Cred}[\mathcal{U}] = \mathsf{nil}$, then abort; else continue. *(Unregistered users cannot successfully sign messages.)*
3. If $\forall i \in [1 \,..\, N] : \mathfrak{a}_i = \varnothing \vee \mathfrak{a}_i \in \mathsf{Cred}[\mathcal{U}][i]$, then:
   (a) Randomly generate ϕ.
   (b) Set $\mathsf{Presented}[\mathfrak{a}, \mu, \phi] \leftarrow \mathsf{true}$.
   (c) Return $(\mathsf{PresentEND}, \phi)$ to $\mathcal{U}$ and $(\mathsf{PresentEND}, \mathfrak{a}, \mu, \phi)$ to $\mathcal{S}$. *(Users with disclosed attribute values successfully sign messages even if they did not call* $\mathsf{Aggregate}$ *before.)*
4. Else abort.

Verify. Upon receiving message $(\mathsf{Verify}, \mathfrak{a}, \mu, \phi)$ from verifier $\mathcal{V}$, do:

1. If $\mathsf{Presented}[\mathfrak{a}, \mu, \phi] \rightarrow \mathsf{false}$, then abort.
2. Else if $\mathsf{Presented}[\mathfrak{a}, \mu, \phi] \rightarrow \mathsf{true}$, then:
   (a) Return $(\mathsf{Verified}, \mathtt{true})$ to $\mathcal{V}$. *(Completeness.)*
   (b) Send $(\mathsf{Verified}, \mathcal{V}, \mathfrak{a}, \mu, \phi, \mathtt{true})$ to $\mathcal{S}$.
3. Else if $\mathsf{Presented}[\mathfrak{a}, \mu, \phi] \rightarrow \mathsf{nil}$, then:
   (a) If there exists a dishonest user $\mathcal{U}'$ with attribute values $\mathfrak{a}' = (\mathfrak{a}'_1, \ldots, \mathfrak{a}'_N)$ s.t. $\forall \mathfrak{a}_i \neq \varnothing$, we either have $\mathfrak{a}_i = \mathfrak{a}'_i$, or have $\mathfrak{a}_i \neq \mathfrak{a}'_i$ but issuer I_i is dishonest, then query $\mathcal{S}$ with $(\mathsf{Verify}, \mathfrak{a}, \mu, \phi)$. On receiving the response $(\mathsf{Verified}, \mathsf{bool})$, do:
      i. Set $\mathsf{Presented}[\mathfrak{a}, \mu, \phi] \leftarrow \mathsf{bool}$.
      ii. Return $(\mathsf{Verified}, \mathsf{bool})$ to $\mathcal{V}$.
   (b) Else:
      i. Set $\mathsf{Presented}[\mathfrak{a}, \mu, \phi] \leftarrow \mathsf{false}$.
      ii. Return $(\mathsf{Verified}, \mathtt{false})$ to $\mathcal{V}$. *(Unforgeability.)*
      iii. Send $(\mathsf{Verified}, \mathcal{V}, \mathfrak{a}, \mu, \phi, \mathtt{false})$ to $\mathcal{S}$.

Fig. 1. Ideal functionality $\mathcal{F}$ for multi-issuer aggregate anonymous credentials

Definition 2 (Binary Relation). *A binary relation $\mathcal{R}$ is defined by a pair $(\mathbb{x}, \mathbb{w})$ where $\mathbb{x}$ is an instance (public input), and $\mathbb{w}$ a witness (private input). If pair $(\mathbb{x}, \mathbb{w})$ satisfies $\mathcal{R}$, then we write $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$; otherwise, $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 0$. We call $\mathcal{L}_{\mathcal{R}} = \{\mathbb{x} : \exists \mathbb{w} \text{ s.t. } \mathcal{R}(\mathbb{x}, \mathbb{w}) = 1\}$ as the language of relation $\mathcal{R}$.*

A SoK for a binary relation $\mathcal{R}$ consists of the following algorithms:

- $\mathsf{SoK.Setup}(\mathcal{R}) \to pp$: on input relation $\mathcal{R}$, output public parameters pp.
- $\mathsf{SoK.Sign}(pp, \mathbb{x}, \mathbb{w}, \mu) \to \phi$: on input public parameters pp, pair $(\mathbb{x}, \mathbb{w})$ such that $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$, and message μ, output a signature of knowledge ϕ.
- $\mathsf{SoK.Verify}(pp, \mathbb{x}, \mu, \phi) \to \{0, 1\}$: on input public parameters pp, public input $\mathbb{x}$, message μ, and signature ϕ, output 0 or 1.

A SoK is said to be secure if it satisfies the properties of correctness, simulatability, and extraction, as defined below.

Correctness. A SoK is correct if and only if for all pairs $(\mathbb{x}, \mathbb{w})$ such that $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$, and for all admissible messages μ, the following holds:

$$\Pr\left[\mathsf{SoK.Verify}(pp, \mathbb{x}, \mu, \phi) = 1 \;\middle|\; \begin{array}{c} \mathcal{R}(\mathbb{x}, \mathbb{w}) = 1 \\ pp \leftarrow \mathsf{SoK.Setup}(\mathcal{R}) \\ \phi \leftarrow \mathsf{SoK.Sign}(pp, \mathbb{x}, \mathbb{w}, \mu) \end{array}\right] = 1.$$

To capture the simulatability and extraction properties, we introduce the following additional algorithms:

- $\mathsf{SoK.SimSetup}(\mathcal{R}) \to (pp, td)$: on input relation $\mathcal{R}$, output public parameters pp and trapdoor td. The distribution of pp is computationally indistinguishable from that generated by $\mathsf{SoK.Setup}(\mathcal{R})$.
- $\mathsf{SoK.SimSign}(pp, td, \mathbb{x}, \mu) \to \phi$: on input public parameters pp, trapdoor td, public input $\mathbb{x}$, and message μ, output simulated signature ϕ (i.e., signatures produced without access to a valid witness).
- $\mathcal{E}(pp, td, \mathbb{x}, \mu, \phi) \to \hat{\mathbb{w}}$: on input public parameters pp, trapdoor td, public input $\mathbb{x}$, message μ, and signature ϕ, output a potential witness $\hat{\mathbb{w}}$.

Simulatability. There exists a polynomial-time simulator consisting of algorithms $\mathsf{SoK.SimSetup}$ and $\mathsf{SoK.SimSign}$ as given above such that for all PPT $\mathcal{A}$, the following holds:

$$\left|\Pr\left[b = 1 \;\middle|\; \begin{array}{c} (pp, td) \leftarrow \mathsf{SoK.SimSetup}(\mathcal{R}) \\ b \leftarrow \mathcal{A}^{\mathcal{S}(pp, td, \cdot, \cdot, \cdot)}(s, pp) \end{array}\right] - \Pr\left[b = 1 \;\middle|\; \begin{array}{c} pp \leftarrow \mathsf{SoK.Setup}(\mathcal{R}) \\ b \leftarrow \mathcal{A}^{\mathsf{SoK.Sign}(pp, \cdot, \cdot, \cdot)}(s, pp) \end{array}\right]\right| \leq \varepsilon(\kappa),$$

where the oracle $\mathcal{S}$ receives the values $(\mathbb{x}, \mathbb{w}, \mu)$ as inputs, checks that $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$, and returns $\phi \leftarrow \mathsf{SoK.SimSign}(pp, td, \mathbb{x}, \mu)$.

Extractability. There exists an extractor algorithm $\mathcal{E}$ such that for all PPT $\mathcal{A}$ there exists a negligible function $\varepsilon(\kappa)$, such that for all auxiliary inputs aux polynomial in κ, we have the following:

$$\Pr\left[\begin{array}{c}\mathcal{R}(\mathbb{x},\hat{\mathbb{w}})=1 \ \vee \ (\mathbb{x},\mu)\in\mathcal{Q}\vee\\ \mathsf{SoK.Verify}(pp,\mathbb{x},\mu,\phi)=0\end{array}\ \middle|\ \begin{array}{c}(pp,td)\leftarrow\mathsf{SoK.SimSetup}(\mathcal{R})\\ (\mathbb{x},\mu,\phi)\leftarrow\mathcal{A}^{\mathcal{S}(pp,td,\cdot,\cdot,\cdot)}(\mathsf{aux},pp)\\ \hat{\mathbb{w}}\leftarrow\mathcal{E}(pp,td,\mathbb{x},\mu,\phi)\end{array}\right]=1-\varepsilon(\kappa),$$

where $\mathcal{Q}$ is the set of prior valid queries answered by $\mathcal{S}(pp,td,\cdot,\cdot,\cdot)$. In other words, $\mathcal{Q}$ contains all $(\mathbb{x},\mu)$ for which $\mathcal{A}$ previously submitted $\mathbb{w}$ with $\mathcal{R}(\mathbb{x},\mathbb{w})=1$ and received a signature ϕ.

3.3 Aggregate Tag-Based Signatures (ATS)

We define an aggregate tag-based signature scheme as one that allows signatures computed over the same tag to be aggregated into a compact signature. An aggregate tag-based signature scheme is defined by the following algorithms:

- $\mathsf{ATS.KeyGen}(1^\kappa)\to(sk,pk)$: On input security parameter 1^κ, output keypair (sk,pk).
- $\mathsf{ATS.Sign}(sk,\tau,m)\to\sigma$: On input secret key sk, tag $\tau\in\{0,1\}^*$, and message $m\in\{0,1\}^*$, output signature σ.
- $\mathsf{ATS.Randomize}(\tau,\sigma)\to\sigma'$: On input tag τ and signature σ, output randomized signature σ'.
- $\mathsf{ATS.Verify}(pk,\tau,m,\sigma)\to\{0,1\}$: on input public key pk, tag τ, message m, and signature σ, return 0 or 1.
- $\mathsf{ATS.Aggregate}(\sigma_1,\ldots,\sigma_n)\to\sigma$: on input signatures $\sigma_1,\ldots,\sigma_n$ (all under the *same* tag τ), output aggregate signature σ.
- $\mathsf{ATS.VerifyAgg}((pk_i)_{i\in[1\,..\,n]},\tau,(m_i)_{i\in[1\,..\,n]},\sigma)\to\{0,1\}$: on input public keys $(pk_i)_{i\in[1\,..\,n]}$, tag τ, messages $(m_i)_{i\in[1\,..\,n]}$, and aggregate signature σ, return 0 or 1.

An aggregate tag-based signature scheme should satisfy correctness and existentially unforgeability, as defined below.

Definition 3 (Correctness). *An ATS scheme satisfies correctness if for all security parameters κ, all $n\geq 1$, all tags $\tau\in\{0,1\}^*$, and all messages $m_i\in\{0,1\}^*$, the following holds.*

Let $\mathbb{I}\subseteq[1\,..\,n]$. For all $i\in\mathbb{I}$, sample $(sk_i,pk_i)\leftarrow\mathsf{ATS.KeyGen}(1^\kappa)$ and let $\sigma_i\leftarrow\mathsf{ATS.Sign}(sk_i,\tau,m_i)$. Then σ is produced as follows:

$$\sigma\leftarrow\mathsf{ATS.Aggregate}((\sigma_i)_{i\in\mathbb{I}}).$$

Then we have the following:

$$\forall i\in\mathbb{I}:\mathsf{ATS.Verify}(pk_i,\tau,m_i,\sigma_i)=1\ \wedge\ \mathsf{ATS.VerifyAgg}\big((pk_i)_{i\in\mathbb{I}},\tau,(m_i)_{i\in\mathbb{I}},\sigma\big)=1.$$

Randomization also preserves validity, meaning that for all $\sigma'\to\mathsf{ATS.Randomize}(\tau,\sigma)$ and $\mathsf{ATS.Verify}(pk,\tau,m,\sigma)=1$, we also have $\mathsf{ATS.Verify}(pk,\tau,m,\sigma')=1$

$\underline{\textsc{corrupt}(pk_i)}$
If $pk_i \in \mathsf{PK}$, then:

- $\mathcal{C} \leftarrow \mathcal{C} \cup \{pk_i\}$;
- return sk_i.

$\underline{\textsc{sign}(pk_i, \tau, m)}$
If $pk_i \in \mathsf{PK}$, then:

- $\Sigma \leftarrow \Sigma \cup \{(pk_i, \tau, m)\}$;
- return $\sigma \leftarrow \mathsf{ATS.Sign}(sk_i, \tau, m)$.

$\underline{\textsc{Experiment for Existential Unforgeability Under Tag-Based Aggregation}}$

- $(\mathbb{I} \subseteq [1 \,..\, n], (pk_i)_{i\in\mathbb{I}}, \tau, (m_i)_{i\in\mathbb{I}}, \sigma) \leftarrow \mathcal{A}^{\textsc{sign},\textsc{corrupt}}(\mathsf{PK})$;
- $b \leftarrow \mathsf{ATS.VerifyAgg}\big((pk_i)_{i\in\mathbb{I}}, \tau, (m_i)_{i\in\mathbb{I}}, \sigma\big)$.

$\mathcal{A}$ wins the experiment if and only if $b = 1 \;\; \wedge \;\; \exists\, i \in \mathbb{I} : pk_i \notin \mathcal{C} \;\; \wedge \;\; (pk_i, \tau, m_i) \notin \Sigma$.

Fig. 2. Definition of the oracles available to the adversary and the experiment for Existential Unforgeability Under Tag-Based Aggregation.

Definition 4 (Existential Unforgeability Under Tag-Based Aggregation). *An ATS scheme* ATS *is existentially unforgeable under tag-based aggregation if for any probabilistic polynomial-time adversary* $\mathcal{A}$*, with access to corruption oracle* CORRUPT *and signing oracle* SIGN*, the probability that* $\mathcal{A}$ *wins the experiment depicted in Fig. 2 is negligible in security parameter* κ*.*

4 Black-Box Construction

In the following, we show how to construct an aggregate anonymous credentials scheme in a black-box manner, using digital signatures, aggregate tag-based signatures and signatures of knowledge. Section 5 describes how to instantiate the various primitives.

Setup. The registration authority RA samples a key pair $(sk, pk) \leftarrow \mathsf{DS.KeyGen}(1^\kappa)$ for the digital signature scheme DS and publishes pk, and generates system-wide public parameters pp for the necessary signatures of knowledge, needed throughout. Each issuer I_i then generates a keypair $(isk_i, ipk_i) \leftarrow \mathsf{ATS.KeyGen}(1^\kappa)$ for the aggregate tag-based signature scheme ATS and publishes ipk_i.

Register. A user $\mathcal{U}$ samples a keypair $(usk, upk) \leftarrow \mathsf{KeyGen}(1^\kappa)$, where KeyGen is a key generation algorithm, and sends public key upk to RA together with a SoK on a fresh nonce μ from RA that proves $\mathcal{U}$ knows the corresponding secret key usk. If the SoK verifies, then RA assigns $\mathcal{U}$ a unique tag τ and issues $\psi \leftarrow \mathsf{DS.Sign}(sk, upk \| \tau)$.

$\mathcal{U}$ initializes their credential state as follows:

$$\mathsf{CRED} = (\tau, usk, upk, \psi, \mathsf{cred}[1], \ldots, \mathsf{cred}[N]),$$

with $\mathsf{cred}[i] = \varnothing$ for all $i \in [N]$.

$\mathsf{IssueCred}$. On input $(\tau, upk, \psi, \mathfrak{a}_i)$ from $\mathcal{U}$, issuer I_i first sets $m = (upk, \tau)$, and checks that $\mathsf{DS.Verify}(pk, m, \psi) = 1$. If not, it rejects. Otherwise, it verifies that $\mathcal{U}$ knows usk via a SoK. If this check succeeds and $\mathcal{U}$ holds attribute value $\mathfrak{a}_i$, then I_i sends back $\sigma_i \leftarrow \mathsf{ATS.Sign}(isk_i, \tau, \mathfrak{a}_i)$, and $\mathcal{U}$, subsequently, sets $\mathsf{cred}[i] = (\mathfrak{a}_i, \sigma_i)$.

$\mathsf{Present}$. Let $\mathsf{CRED} = (\tau, usk, upk, \psi, \mathsf{cred}[1], \ldots, \mathsf{cred}[N])$, where $\mathsf{cred}[i] = (\mathfrak{a}_i, \sigma_i)$. To prove possession of attribute values $(\mathfrak{a}_i)_{i\in\mathbb{I}}$ to some verifier $\mathcal{V}$ that provides a fresh nonce μ, user $\mathcal{U}$ computes $\sigma \leftarrow \mathsf{ATS.Aggregate}(\{\sigma_i\}_{i\in\mathbb{I}})$, and SoK $\phi \leftarrow \mathsf{SoK.Sign}(pp, \mathbb{x}, \mathbb{w}, \mu)$, where $\mathbb{x} = (pk, \mathbb{I}, \{ipk_i, \mathfrak{a}_i\}_{i\in\mathbb{I}})$ and $\mathbb{w} = (\tau, usk, upk, \psi, \sigma)$, and which proves the following:

$$(usk, upk) \leftarrow \mathsf{KeyGen}(1^\kappa) \quad \wedge \quad \mathsf{DS.Verify}(pk, upk \| \tau, \psi) = 1$$
$$\wedge \quad \mathsf{ATS.VerifyAgg}(\{ipk_i\}_{i\in\mathbb{I}}, \tau, \{\mathfrak{a}_i\}_{i\in\mathbb{I}}, \sigma) = 1.$$

Verify. On input $(\{\mathfrak{a}_i\}_{i\in\mathbb{I}}, \mu, \phi)$, verifier $\mathcal{V}$ returns the output of $\mathsf{SoK.Verify}(pp, \mathbb{x}, \mu, \phi)$, with $\mathbb{x} = (pk, \mathbb{I}, \{ipk_i, \mathfrak{a}_i\}_{i\in\mathbb{I}})$.

Theorem 1. *If* DS *is an existentially unforgeable digital signature scheme,* ATS *an existentially unforgeable aggregate tag-based signature scheme under tag-based aggregation, and* SoK *a secure signature of knowledge, then the above protocol securely realizes ideal functionality* $\mathcal{F}$ *in the standalone model.*

Due to space limitations, the proof of Theorem 1 is deferred to Appendix A.

5 Instantiation

Notation. Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ be cyclic groups of prime order p, equipped with an efficiently computable, non-degenerate bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.

We denote elements in $\mathbb{G}_1$ by upper-case letters, and elements in $\mathbb{G}_2$ by upper-case letters with hats. Elements in $\mathbb{Z}_p$ are denoted by lower-case letters. Let P and $\widehat{Q} \in \mathbb{G}_2$ denote two random generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. Let $\mathbf{H} : \{0,1\}^* \to \mathbb{G}_1$ and $\mathbf{G} : \{0,1\}^* \to \mathbb{G}_1$ be two cryptographic hash functions.

Overview. In this instantiation, each user $\mathcal{U}$ has a pair of secret and public keys $(usk, upk = P^{usk}) \in \mathbb{Z}_p \times \mathbb{G}_1$. The digital signature DS is leverages AGHO structure preserving signatures [1], which are used to sign a pair $(upk, \mathbf{G}(\tau)) \in \mathbb{G}_1 \times \mathbb{G}_1$, where upk is the user's public key and τ their unique tag (cf. Section 5.1). The aggregate tag-based signature ATS is instantiated with our own construction detailed in Section 5.2. The SoK is instantiated by applying the Fiat-Shamir heuristic to a Sigma protocol as described in Appendix C.

5.1 AGHO Structure-Preserving Signature Scheme

The AGHO scheme supports signing vectors of elements in $\mathbb{G}_1$, $\mathbb{G}_2$, or both. In the following, we only describe signing elements of $\mathbb{G}_1$. This scheme is made up of the following algorithms:

- $\mathsf{AGHO.KeyGen}(1^\kappa, l) \rightarrow (sk, pk)$: On input a security parameter κ and vector length l, sample $(u_1, \ldots, u_l, v) \xleftarrow{\$} \mathbb{Z}_p^{l+1}$ uniformly at random. Set the secret key $sk = (u_1, \ldots, u_l, v)$ and the public key $pk = (\widehat{U}_1, \ldots, \widehat{U}_l, V)$ with $\widehat{U}_i = \widehat{Q}^{u_i} \in \mathbb{G}_2$ and $V = P^v \in \mathbb{G}_1$. Finally, output (sk, pk).
- $\mathsf{AGHO.Sign}(sk, \boldsymbol{M}) \rightarrow \psi$: On input the secret key $sk = (u_1, \ldots, u_l, v)$ and a message vector $\boldsymbol{M} = (M_1, \ldots, M_l) \in \mathbb{G}_1^l$, sample $r \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random and compute the signature as follows:

$$\sigma = (\widehat{R}, \widehat{S}, T) = (\widehat{Q}^r, \widehat{R}^v, (P \cdot \prod_{i=1}^{l} M_i^{-u_i})^{1/r}).$$

- $\mathsf{AGHO.Randomize}(\psi) \rightarrow \psi'$: On input signature $\psi = (\widehat{R}, \widehat{S}, T)$, randomly pick $r' \leftarrow \mathbb{Z}_p$ and output randomized signature formed as follows:

$$\psi' = (\widehat{R}', \widehat{S}', T') = (\widehat{R}^{r'}, \widehat{S}^{r'}, T^{\frac{1}{r'}}).$$

- $\mathsf{AGHO.Verify}(pk, \boldsymbol{M}, \psi) \rightarrow \{0, 1\}$: On input a public key pk, a message vector $\boldsymbol{M} = (M_1, \ldots, M_l) \in \mathbb{G}_1^l$, and a signature $\psi = (\widehat{R}, \widehat{S}, T)$, output 1 if both of the following pairing equations hold:

$$e(V, \widehat{R}) = e(P, \widehat{S}) \quad \text{and} \quad e(T, \widehat{R}) \cdot \prod_{i=1}^{l} e(M_i, \widehat{U}_i) = e(P, \widehat{Q}).$$

  Otherwise output 0.

AGHO is existentially unforgeable under chosen-message attacks under the q-Strong Diffie-Hellman (q-SDH) assumption.

Definition 5. *q-SDH assumption. Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ be bilinear groups of prime order p with generators $P \in \mathbb{G}_1$ and $\widehat{Q} \in \mathbb{G}_2$. The q-SDH assumption states that, given the following:*

$$(P, P^x, P^{x^2}, \ldots, P^{x^q}, \widehat{Q}),$$

for a randomly chosen $x \xleftarrow{\$} \mathbb{Z}_p$, it is computationally infeasible for any PPT adversary to output a pair $(c, P^{1/(x+c)})$ for any $c \in \mathbb{Z}_p \setminus \{-x\}$.

Now we show how registration authority RA uses AGHO signatures to implement digital signature DS.

- $\mathsf{DS.KeyGen}(1^\kappa) \rightarrow (sk, pk)$: It calls $\mathsf{AGHO.KeyGen}(1^\kappa, 2)$ to produce a key pair for signing two elements.
- $\mathsf{DS.Sign}(sk, m) \rightarrow \psi$: It first parse m as $upk \| \tau$, computes $\Gamma = \mathbf{G}(\tau)$, and outputs signature:

$$\psi \leftarrow \mathsf{AGHO.Sign}(sk, (upk, \Gamma)).$$

- $\mathsf{DS.Verify}(pk, m, \psi) \rightarrow \{0, 1\}$: It first parses m as $upk \| \tau$, computes $\Gamma = \mathbf{G}(\tau)$, and returns bit b defined as:

$$b \leftarrow \mathsf{AGHO.Verify}(pk, (upk, \Gamma), \psi).$$

5.2 Our Tag-Based Aggregate Signature (ATS)

Our tag-based aggregate signature scheme is an enhanced aggregate BLS signature scheme [7,6] that signs pairs of messages and tags. The key differences are tag-restricted aggregation and ability to randomize signatures. Our scheme is comprised of the following algorithms:

- $\mathsf{ATS.KeyGen}(1^\kappa) \rightarrow (sk, pk)$: Randomly select $x \in \mathbb{Z}_p$, compute $\widehat{X} = \widehat{Q}^x \in \mathbb{G}_2$, and output key pair $(sk, pk) = (x, \widehat{X})$.
- $\mathsf{ATS.Sign}(sk, \tau, m) \rightarrow \sigma$: On input secret key $sk = x \in \mathbb{Z}_p$, and pair of tag and message $(\tau, m) \in \{0,1\}^* \times \{0,1\}^*$, randomly select $s \in \mathbb{Z}_p$, and output signature $\sigma = (\widehat{A}, B) \in \mathbb{G}_2 \times \mathbb{G}_1$ constructed as follows:

$$\sigma = (\widehat{Q}^s, \mathbf{H}(m)^x \cdot \mathbf{G}(\tau)^s).$$

- $\mathsf{ATS.Randomize}(\tau, \sigma) \rightarrow \sigma'$: On input tag τ, signature $\sigma = (\widehat{A}, B)$, randomly select $s' \in \mathbb{Z}_p$ and output randomized signature formed as follows:

$$\sigma' = (\widehat{A}', B') = (\widehat{A} \cdot \widehat{Q}^{s'}, B \cdot \mathbf{G}(\tau)^{s'}).$$

- $\mathsf{ATS.Verify}(pk, \tau, m, \sigma) \rightarrow 0/1$: On input public key pk, tag $\tau \in \{0,1\}^*$, message $m \in \{0,1\}^*$, and signature $\sigma = (\widehat{A}, B)$, output 1 if the following equation is satisfied:

$$e(B, \widehat{Q}) = e(\mathbf{H}(m), pk) \cdot e(\mathbf{G}(\tau), \widehat{A}).$$

  Otherwise, output 0.
- $\mathsf{ATS.Aggregate}(\{\sigma_i\}_{i \in [1..n]}) \rightarrow \sigma$: On input signatures $\sigma_i = (\widehat{A}_i, B_i)$, output

$$\sigma = (\widehat{A}, B) = (\prod_{i=1}^{n} \widehat{A}_i, \prod_{i=1}^{n} B_i).$$

- $\mathsf{ATS.VerifyAgg}(\{pk_i\}_{i \in [1..n]}, \tau, \{m_i\}_{i \in [1..n]}, \sigma) \rightarrow 0/1$: On input public keys $\{pk_i\}_{i \in [1..n]}$, tag τ, messages $\{m_i\}_{i \in [1..n]}$, and aggregate signature σ, parse $\sigma = (\widehat{A}, B)$ and output 1 if the following holds:

$$e(B, \widehat{Q}) = e(\mathbf{G}(\tau), \widehat{A}) \cdot \prod_{i=1}^{n} e(\mathbf{H}(m_i), pk_i).$$

  Otherwise, output 0.

This signature is existentially unforgeable under tag-based aggregation in the random oracle model under the Bilinear Computational Diffie-Hellman (BCDH) assumption and a modified variant of the same assumption. To the best of our knowledge, the modified BCDH assumption has not been explicitly introduced in prior work. However, it fits within the Uber assumption framework and can be shown to hold in the generic bilinear group model [4,9].

Definition 6 (The BCDH Assumption). *We say that the BCDH assumption holds if given* $(P, P^x, P^y, P^z, \widehat{Q}, \widehat{Q}^x, \widehat{Q}^y, \widehat{Q}^z)$*, it is computationally infeasible for any PPT algorithm to output* $e(P, \widehat{Q})^{xyz}$*.*

Definition 7 (The Modified BCDH Assumption). *We say that the modified BCDH (m-BCDH) assumption holds if given* $(P, P^{1/y}, \widehat{Q}, \widehat{Q}^x, \widehat{Q}^y)$*, it is computationally infeasible for any PPT algorithm to output* $e(P, \widehat{Q})^{xy}$*.*

Theorem 2. *If the BCDH and m-BCDH assumptions hold, then our tag-based aggregate signature is existentially unforgeable under tag-based aggregation, in the random oracle model.*

The proof of this theorem can be found in Appendix B.

5.3 Signatures of Knowledge

The user proves that she possesses (i) a valid registration signature, (ii) an aggregate credential under a single tag, and (iii) the secret key corresponding to the registered public key corresponding to the given tag, without revealing either the tag or the public key.

Let $pp = (P, \widehat{Q})$; $\mathbb{x} = (pk, \{\mathfrak{a}_i, ipk_i\}_{i\in\mathbb{I}})$ with $pk = (\widehat{U}_1, \widehat{U}_2, V)$ and $ipk_i = \widehat{X}_i$. Let $\mathbb{w} = (usk, upk, \Gamma = \mathbf{G}(\tau), \sigma, \psi)$ with $\sigma = (\widehat{A}, B)$ and $\psi = (\widehat{R}, \widehat{S}, T)$.

To present set of attribute values $\{\mathfrak{a}_i\}_{i\in\mathbb{I}}$ after receiving a nonce μ from verifier $\mathcal{V}$, user $\mathcal{U}$ aggregates the corresponding tag-based signatures and produces a SoK for μ with respect to relation $\mathcal{R}$ defined as $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$, if and only if, the following equalities hold:

$$\begin{aligned} e(V, \widehat{R}) = e(P, \widehat{S}) \quad ; \quad e(T, \widehat{R}) \cdot e(P^{usk}, \widehat{U}_1) \cdot e(\Gamma, \widehat{U}_2) = e(P, \widehat{Q}) \ ; \\ upk = P^{usk} \quad ; \quad e(B, \widehat{Q}) = e(\Gamma, \widehat{A}) \cdot \prod_{i\in\mathbb{I}} e(\mathbf{H}(\mathfrak{a}_i), \widehat{X}_i). \end{aligned} \tag{1}$$

Since we assume that the registration authority only issues signatures on well-formed messages, i.e., pairs $(upk, \mathbf{G}(\tau)) \in \mathbb{G}_1 \times \mathbb{G}_1$ for some $\tau \in \mathbb{Z}_p$, we do not require the user to prove knowledge of the pre-image τ such that $\Gamma = \mathbf{G}(\tau)$, resulting in a more efficient SoK.

To reduce the computational cost incurred by this SoK, we randomize the signatures and equations in a way that removes the need for producing zero-knowledge proofs over pairing equations in $\mathbb{G}_t$, and relies on zero-knowledge proofs of knowledge and equality of discrete logarithms of group elements in $\mathbb{G}_1$.

More specifically, user $\mathcal{U}$ first randomizes signatures $\sigma' \leftarrow \mathsf{ATS.Randomize}(\sigma)$ and $\psi' \leftarrow \mathsf{AGHO.Randomize}(\psi)$. Let $\sigma' = (\widehat{A}', B')$ and $\psi' = (\widehat{R}', \widehat{S}', T')$. The user then samples a random $\delta \in \mathbb{Z}_p^*$ and computes the following:

$$\begin{gathered} \forall i \in \mathbb{I}: \quad H_i = \mathbf{H}(\mathfrak{a}_i)^\delta \quad ; \quad \bar{P} = P^\delta \quad ; \quad \bar{T} = T'^\delta \quad ; \quad \bar{B} = B'^\delta \ ; \\ \bar{\Gamma} = \Gamma^\delta \quad ; \quad \bar{V} = V^\delta \quad ; \quad \bar{upk} = upk^\delta. \end{gathered}$$

Let $pp = P$; $\mathbb{x} = (\bar{upk}, \{\mathfrak{a}_i, H_i\}_{i\in\mathbb{I}}, \bar{P})$; $\mathbb{w} = (\delta, usk)$, and let $\mathcal{R}$ be the relation defined as $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$, if and only if, the following holds:

$$\bar{P} = P^{\delta} \quad ; \quad \bar{V} = V^{\delta} \quad ; \quad H_i = \mathbf{H}(\mathfrak{a}_i)^{\delta} \quad ; \quad \bar{upk} = \bar{P}^{usk}. \tag{2}$$

Next, $\mathcal{U}$ produces η, a SoK for $\mathcal{R}$, and defines presentation ϕ as

$$\phi = (\eta, \bar{P}, \bar{V}, \widehat{R}', \widehat{S}', \bar{T}, \widehat{A}', \bar{B}, \bar{\Gamma}, \bar{upk}, \{\mathfrak{a}_i, H_i\}_{i\in\mathbb{I}}).$$

Verifier $\mathcal{V}$ accepts ϕ if and only if, $\bar{P} \neq 1$, η is a valid SoK for the relation depicted in equation 2, and the following equalities hold:

$$e(\bar{B}, \widehat{Q}) = \prod_{i\in\mathbb{I}} e(H_i, \widehat{X}_i) \cdot e(\bar{\Gamma}, \widehat{A}') \ ;$$
$$e(\bar{P}, \widehat{Q}) = e(\bar{T}, \widehat{R}') \cdot e(\bar{upk}, \widehat{U}_1) \cdot e(\bar{\Gamma}, \widehat{U}_2) \ ;$$
$$e(\bar{V}, \widehat{R}') = e(\bar{P}, \widehat{S}').$$

In Appendix C, we describe how to compute η and show that (1) if $\mathcal{U}$ produces a valid ϕ, then a witness for the relation depicted in Equation 1 can be extracted, and (2) ϕ is simulatable under the decisional Diffie-Hellman (DDH) assumption in $\mathbb{G}_1$.

Definition 8 (DDH Assumption). *Let $\mathbb{G}$ be a cyclic group of large prime order p and P be a generator. We say that the DDH assumption holds in $\mathbb{G}$ if given a triple (P^x, P^y, P^z), it is computationally infeasible to tell whether $z = xy$ or not.*

5.4 Extensions

The construction supports also several straightforward extensions.

Supporting New Issuers. New issuers can join the system without impacting the construction. In fact, the aggregation of credentials and their presentation is agnostic to the number of issuers.

Increasing Attributes per Issuer. The current construction assumes that each issuer produces a credential for a single attribute type. The current construction captures settings where issuers generate credentials for multiple attribute types by having each issuer hold different public key for each attribute type. Alternatively, if it is desirable to have the issuer hold a single public key regardless of the number of attribute types it is responsible for, then we modify the way the issuer signs. Instead of signing pair $(\tau, \mathfrak{a})$, the issuer signs $(\tau, \mathtt{att}\|\mathfrak{a})$, with $\mathtt{att}$ the attribute type and $\mathfrak{a}$ the attribute value.

Epoch-Based Revocation. The construction can be extended to support epoch-based revocation, in the following way. Instead of signing pair $(upk, \mathbf{G}(\tau))$, registration authority RA signs triple $(upk, \mathbf{G}(\tau), P^e)$, where e identifies the current epoch. Issuers, on their end, sign pair $(\tau, e \| \mathtt{att} \| \mathfrak{a})$, instead of pair $(\tau, \mathtt{att} \| \mathfrak{a})$; this ties the attribute credential to epoch e. During credential presentation, users prove that they hold a valid signature from RA and valid aggregate tag-based signatures from the relevant issuers, for the current epoch.

We leave the study of accumulator-based revocation and the revocation of individual attribute credentials to future work.

6 Evaluation

We evaluate the performance of our credential system instantiated over pairing-friendly groups. Concretely, we implement the scheme using the BLS12-381 pairing, where group elements in $\mathbb{G}_1$ are defined over the base field $\mathbb{Z}_p$ and group elements in $\mathbb{G}_2$ are defined over the quadratic extension field $\mathbb{Z}_p^2$. All cryptographic operations are performed in the source groups $\mathbb{G}_1$, $\mathbb{G}_2$, and the target group $\mathbb{G}_T$ induced by the bilinear pairing; our construction does not rely on curve-specific properties beyond bilinearity and can be instantiated with other pairing-friendly curves.

We implement the credential system in Go using the `gnark-crypto` library [8], which provides optimized implementations of group and pairing operations for BLS12-381. Benchmarks were collected on a MacBook Pro with an M1 Pro chip and 16GB of RAM.

To assess efficiency, we benchmark the cost of Present and Verify while varying (i) the total number of attributes n, and (ii) the number of selectively revealed attributes k. Figures 3 and 4 display the results.

Our Present procedure consists of two phases: setup captures user registration and credential issuance for all n attributes, which is independent of k, and proving captures the user generating proofs corresponding to the k revealed attributes. We report these phases separately, as they capture distinct costs: setup grows with n, while proving grows with k. Verification also only depends on the number of revealed attributes.

Even for $n = 128$ attributes with $k = 64$ selectively revealed, both proving and verification complete in under 10ms on a laptop. Setup scales linearly with n, while proving and verification scale linearly with k. This contrasts with credential systems such as Idemix [15], where proving and presentation costs depend on the total number of attributes, rather than the number of disclosed attributes. Overall, our evaluation demonstrates the practical efficiency of the system: setup and presentation costs are low enough for client devices, and verification remains lightweight.

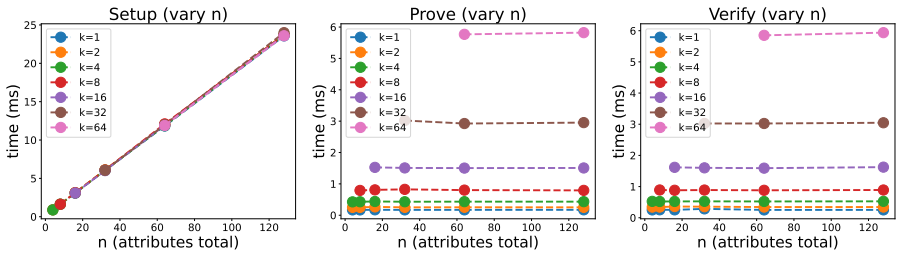

Fig. 3. Scaling of setup, proving, and verification with total attributes n varying, for fixed k.

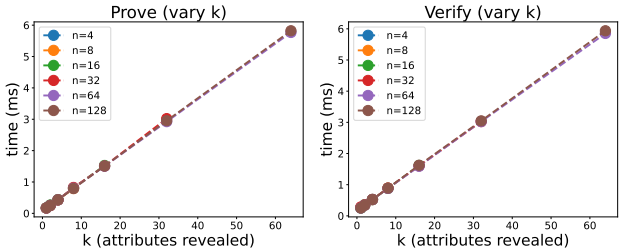

Fig. 4. Scaling of setup, proving, and verification with revealed attributes k varying, for fixed n.

7 Related Work

Single-Issuer Anonymous Credential Systems. Early work by Camenisch and Lysyanskaya introduced anonymous credential systems based on the strong-RSA assumption [12,13], enabling issuance of signatures on committed values. They later proposed a pairing-based construction in the random oracle model [14], though signature size and verification complexity grow linearly with the number of attribute values signed. An extension [22] allows for a constrained form of aggregation, but only in settings where all values are known ahead of time.

Further improvements targeted efficiency. Pointcheval and Sanders [23] introduced randomizable signatures, with unforgeability proven in the generic bilinear group model. Their scheme supports sequential aggregation. These systems operate in a centralized, single-issuer setting where one authority certifies all attributes. More recently, structure-preserving signatures on equivalence classes were constructed [20]. These facilitate very efficient privacy-preserving proofs. The security of the scheme is proven in the generic group model. Unlinkable proofs of knowledge over a subset of signed messages, building upon the Pointcheval-Sanders signature scheme have also been constructed [24]. Their security is also proven in the generic group model.

BBS+ credentials support selective disclosure in a single-issuer setting with verification linear in the number of revealed attributes, and form the basis of the ongoing IETF standardization effort [5,2].

Multi-Issuer Anonymous Credentials. Some schemes are designed to decentralize the issuer, by delegating issuance to sub-issuers or by distributing issuance responsibilities amongst several issuers [3,10,25]. In these cases, the now decentralized issuer is still responsible for all attributes. While they reduce the trust in a central authority, they do not fully reflect real-world scenarios where different authorities independently certify distinct attributes.

Aggregate Anonymous Credentials. Aggregate credentials aim to combine independently issued credentials into a single compact credential, while maintaining user privacy during presentations. Canard and Lescuyer introduced indexed aggregate signatures supporting anonymous credentials, with unforgeability under the q-Asymmetric Double Hidden Strong Diffie-Hellman assumption and anonymity under DDH in the random oracle model [16]. Later work achieves efficient unlinkable credential presentations over subsets of messages for multiple issuers, but relies on relatively strong cryptographic assumptions and requires a trusted setup [11]. Hébant and Pointcheval proposed a traceable aggregate signature scheme using randomizable public keys as issuance tags. Their system offers traceability, and supports a single signature per issuer per index, which requires issuers to be stateful [21].

Our Contribution. In contrast to classical anonymous credential schemes, we explicitly consider a decentralized setting in which each attribute is certified by a distinct issuer, and credentials issued by different authorities are combined into a single compact one. Aggregation is performed locally by the user without interaction between issuers and without requiring the issuers to be stateful. We note that the scheme supports selective disclosure of attributes values, and predicate proofs can be supported generically via zk-SNARKs. This would come at a cost to efficiency, and efficient aggregation of predicate proofs is left as future work.

8 Conclusion

We introduced what we call *aggregate tag-based signatures* that enable a user to aggregate signatures produced by different signers but computed under the same tag, in such a way that yields constant-size signatures with efficient verification. Using this primitive, we designed an aggregate anonymous credentials system and proved its security. Our scheme supports selective disclosure of attribute values, and we leave developing efficient aggregate predicate proofs, along with efficient compatible blind issuance protocols, as directions for future work. Our benchmark results demonstrate practical performances, highlighting the scheme's potential for real-world decentralized identity applications.

References

1. Masayuki Abe, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Optimal structure-preserving signatures in asymmetric bilinear groups. In *Advances in Cryptology–CRYPTO 2011: 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings 31*, pages 649–666. Springer, 2011.
2. Man Ho Au, Willy Susilo, and Yi Mu. Constant-size dynamic k-TAA. In Roberto De Prisco and Moti Yung, editors, *Security and Cryptography for Networks*, pages 111–125, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
3. Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009*, pages 108–125, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
4. Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, Lecture Notes in Computer Science, pages 440–456. Springer, 2005.
5. Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matt Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, pages 41–55, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
6. Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432. Springer, 2003.
7. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, 2001.
8. Gautam Botrel, Thomas Piellard, Youssef El Housni, Arya Tabaie, Gus Gutoski, and Ivo Kubjas. Consensys/gnark-crypto: v0.15.0, January 2025.
9. Xavier Boyen. The uber-assumption family. In Steven D. Galbraith and Kenneth G. Paterson, editors, *Pairing-Based Cryptography - Pairing 2008, Second International Conference, Egham, UK, September 1-3, 2008. Proceedings*, Lecture Notes in Computer Science, pages 39–56. Springer, 2008.
10. Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. Practical uc-secure delegatable credentials with attributes and their application to blockchain. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 683–699, New York, NY, USA, 2017. Association for Computing Machinery.
11. Jan Camenisch, Maria Dubovitskaya, Kristiyan Haralambiev, and Markulf Kohlweiss. Composable and modular anonymous credentials: Definitions and practical constructions. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 262–288. Springer, 2015.

12. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT*, pages 93–118, 2001.
13. Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In *Security in Communication Networks: Third International Conference, SCN 2002 Amalfi, Italy, September 11–13, 2002 Revised Papers 3*, pages 268–289. Springer, 2003.
14. Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matt Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, pages 56–72, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
15. Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In *Proceedings of the 9th ACM conference on Computer and communications security*, CCS '02, pages 21–30. ACM, 2002.
16. Sébastien Canard and Roch Lescuyer. Anonymous credentials from (indexed) aggregate signatures. In *Proceedings of the 7th ACM workshop on Digital identity management*, pages 53–62, 2011.
17. Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*, pages 78–96. Springer, 2006.
18. David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.
19. Elizabeth C. Crites and Anna Lysyanskaya. Delegatable anonymous credentials from mercurial signatures. In Mitsuru Matsui, editor, *Topics in Cryptology – CT-RSA 2019*, pages 535–555, Cham, 2019. Springer International Publishing.
20. Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. *Journal of Cryptology*, 32:498–546, 2019.
21. Chloé Hébant and David Pointcheval. Traceable constant-size multi-authority credentials. In *SCN 2022-13th conference on security and cryptography for networks*, volume 13409, pages 411–434. Springer International Publishing, 2022.
22. Kwangsu Lee, Dong Hoon Lee, and Moti Yung. Aggregating cl-signatures revisited: Extended functionality and better efficiency. In *Financial Cryptography and Data Security: 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers 17*, pages 171–188. Springer, 2013.
23. David Pointcheval and Olivier Sanders. Short randomizable signatures. In *Cryptographers' Track at the RSA Conference*, pages 111–126. Springer, 2016.
24. Olivier Sanders. Efficient redactable signature and application to anonymous credentials. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *Public-Key Cryptography - PKC 2020 - 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4-7, 2020, Proceedings, Part II*, volume 12111 of *Lecture Notes in Computer Science*, pages 628–656. Springer, 2020.
25. Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, and George Danezis. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019.

A Proof of Theorem 1

Proof. In the following, we show that the view of a real-world adversary $\mathcal{A}$ during an execution of Π is indistinguishable from the view of an ideal-world simulator $\mathcal{S}$ that interacts with system participants through ideal functionality $\mathcal{F}$. We define a sequence of indistinguishable games that start with Game 0 representing the real world, and end with Game 8 representing the ideal world.

Game 0. This is the real-world execution of protocol Π.

Game 1. The simulator $\mathcal{S}$ executes real-world protocol Π on behalf of the honest parties. That is, $\mathcal{S}$ is provided with the inputs (including secrets) of the honest parties, and at their behest, interacts with corrupt parties (i.e., adversary $\mathcal{A}$), in the real-world, following Π's specifications, and returns the corresponding output. Game 1 is by construction indistinguishable from Game 0.

Game 2. We introduce dummy functionality $\mathcal{F}'$ that forwards the inputs from the honest parties to $\mathcal{S}$ and returns the respective outputs. This modification is syntactic.

Game 3. In this game, $\mathcal{F}'$ handles Register calls from honest users following the description of ideal functionality $\mathcal{F}$. $\mathcal{S}$ first assigns each honest user $\mathcal{U}_i$ a pair of user secret and public keys (usk_i, upk_i). Now, when an honest user $\mathcal{U}$ calls $\mathcal{F}'$ to register, $\mathcal{S}$ learns the user's identity. Given this information, $\mathcal{S}$ retrieves the corresponding pair of secret and public keys (usk, upk) and contacts registration authority RA to register $\mathcal{U}$. $\mathcal{S}$, as a result receives, a unique tag τ and a signature $\psi \leftarrow \mathsf{DS.Sign}(sk, upk \| \tau)$ and stores entry $\langle \mathcal{U}, \tau, upk, \psi \rangle$ (we recall sk is RA's signing key).

Since RA is honest, RA will register users only once and reject duplicate requests, same as $\mathcal{F}'$. In other words, when $\mathcal{F}'$ accepts a registration request, so will RA in the real world, and vice versa. Hence, this game is indistinguishable from Game 2.

Game 4. In this game, $\mathcal{F}'$ also handles credential issuance calls from honest users following the description of ideal functionality $\mathcal{F}$. When $\mathcal{F}'$ receives issuance request $(\mathsf{Issue}, \mathtt{att}_i, \mathfrak{a}_i)$ from honest user $\mathcal{U}$, $\mathcal{S}$ learns the triple $(\mathcal{U}, \mathtt{att}_i, \mathfrak{a}_i)$. To simulate the corresponding credential issuance for $\mathcal{U}$ in the real world, $\mathcal{S}$ retrieves $(usk, upk_)$ and (τ, ψ). If the pair (τ, ψ) for $\mathcal{U}$ does not exist (i.e., $\mathcal{U}$ did not call Register before), then $\mathcal{S}$ abort; otherwise, it honestly executes Π with issuer I_i. There are two cases to consider.

1. I_i is honest. If this is the first time $\mathcal{S}$ simulates $\mathcal{U}$ for I_i, then as per Π's specification, $\mathcal{S}$ will receive signature $\sigma_i \leftarrow \mathsf{ATS.Sign}(isk_i, \tau, \mathfrak{a}_i)$, and record entry $\langle \mathcal{U}, \tau, \mathfrak{a}_i, \sigma_i \rangle$. Otherwise, I_i will reject the credential issuance request.
2. I_i is corrupt. If the execution of Π terminates with $\mathcal{S}$ receiving signature $\sigma_i : \mathsf{ATS.Verify}(ipk_i, \tau, \mathfrak{a}_i, \sigma_i) \rightarrow 1$, then $\mathcal{S}$ records entry $\langle \mathcal{U}, \tau, \mathfrak{a}_i, \sigma_i \rangle$, and sends Accept.

We recall that, in the real world, an honest issuer rejects a credential issuance request from a registered user if a preceding request from the same user has already been processed. Otherwise, the honest issuer accepts and issues the credential. It is easy to see that, in this case, the output of the issuer matches that of $\mathcal{F}'$ if the latter processes the credential request instead. We also recall that when $\mathcal{F}'$ receives a credential issuance request involving a dishonest issuer, $\mathcal{F}'$ follows $\mathcal{S}$ instructions. That is, if $\mathcal{S}$ sends Accept, then $\mathcal{F}'$ accepts the request and updates the credential map accordingly; otherwise, $\mathcal{F}'$ just rejects. Since $\mathcal{S}$ sends Accept only if the dishonest issuer outputs a valid aggregate signature, then the output of the dishonest issuer in the real world is equivalent to the output of $\mathcal{F}'$. Therefore, this game is indistinguishable from Game 4 as long as $\mathcal{S}$ does not abort. Since $\mathcal{U}$ is honest, she will always register first before requesting credentials from the issuers, and hence, the abort event will never occur.

Game 5. During this game, $\mathcal{F}'$ executes aggregate requests from honest users following the description of $\mathcal{F}$. This game is indistinguishable from Game 4, as credential aggregation, in the real world, is a local operation during which adversary $\mathcal{A}$ observes nothing, and therefore, $\mathcal{S}$ need not do anything.

Game 6. In this game, $\mathcal{F}'$ takes charge of signature requests from honest users, following the specifications of $\mathcal{F}$, and $\mathcal{S}$ successfully simulates their executions of Π in the real world thanks to the simulatability of the signature of knowledge SoK. Namely, given message $(\mathsf{PresentEND}, \mathfrak{a}, \mu, \phi)$ from $\mathcal{F}'$ for $\mathfrak{a} = (\mathfrak{a}_1, \ldots, \mathfrak{a}_N)$, $\mathcal{S}$ defines $\mathbb{x} = (pk, (ipk_i, \mathfrak{a}_i)_{i \in \mathbb{I}})$ such that $\forall i \in \mathbb{I} : \mathfrak{a}_i \neq \varnothing$, and outputs $\phi^\star \leftarrow \mathsf{SoK.SimSign}(td, \mathbb{x}, \mu, pp)$, which yields $\mathsf{SoK.Verify}(\mathbb{x}, \phi^\star, pp)$ to return `true`.

Game 7. In this game, $\mathcal{F}'$ processes verification requests from honest verifiers following the description of $\mathcal{F}$, whereas $\mathcal{S}$ simulates the honest verifiers in the real world. Namely, upon receiving a signature in the real world, $\mathcal{S}$ translates the signature into calls to $\mathcal{F}'$, and instead of returning the output of SoK.Verify, it returns the verification result received from $\mathcal{F}'$. In the following, we describe $\mathcal{S}$'s simulation.

Registration. For each registration request upk that a corrupt user $\mathcal{U}$ sends, in the real world, to the honest registration authority RA, $\mathcal{S}$ (acting as RA) first checks if $\mathcal{U}$ knows the secret key usk underlying upk. If not, then $\mathcal{S}$ stops. Otherwise, it executes the rest of the registration protocol, records entry $\langle \mathcal{U}, \tau, upk, \psi \rangle$, and then (acting as a user) transmits the corresponding registration request Register to $\mathcal{F}'$ which processes it in a similar fashion to $\mathcal{F}$.

As per the specifications of Π, the simulated RA in the real world will not accept a registration request that will be rejected by $\mathcal{F}'$, and vice versa. Therefore the output of Π and the output of $\mathcal{F}'$ when processing registration requests is indistinguishable.

Credential Issuance. For each issuance request $(\tau, upk, \psi, \mathfrak{a}_i)$ that a corrupt user $\mathcal{U}$ sends, in the real world, to an honest issuer I_i (simulated by $\mathcal{S}$), $\mathcal{S}$ (acting as $\mathcal{U}$) sends $(\mathsf{Issue}, \mathtt{att}_i, \mathfrak{a}_i)$ to $\mathcal{F}'$. We recall that if $\mathcal{U}$ is not registered or if this is not the first time that $\mathcal{U}$ requests a credential, $\mathcal{F}'$ rejects. Otherwise,

$\mathcal{F}'$ issues the credential. Similarly, in the real world, the simulated I_i returns $\sigma_i \leftarrow \mathsf{ATS.Sign}(isk_i, \tau, \mathfrak{a}_i)$ if the same conditions are met. Hence, the output of Π and the output of $\mathcal{F}'$ when processing credential issuance is indistinguishable. After each successful credential issuance, $\mathcal{S}$ stores entry $\langle \mathcal{U}, \tau, \mathtt{att}_i, \mathfrak{a}_i, \sigma_i \rangle$.

Signature Verification. For each signature $(\boldsymbol{\mathfrak{a}} = (\mathfrak{a}_1, ..., \mathfrak{a}_N), \mu, \phi)$ that a user $\mathcal{U}$ presents, in the real world, to an honest verifier V, $\mathcal{S}$ (simulating V) calls $\mathsf{bool} \leftarrow \mathsf{SoK.Verify}(pp, \boldsymbol{\mathfrak{a}}, \mu, \phi)$. We recall that $\mathfrak{a}_i = \varnothing$ if the attribute value is not disclosed.

If $\mathsf{bool} = \mathtt{false}$, $\mathcal{S}$ sends $(\mu, \boldsymbol{\mathfrak{a}} = (\mathfrak{a}_1, ..., \mathfrak{a}_N), \phi)$ to $\mathcal{F}'$, which returns $(\mathsf{Verified}, \mathsf{bool}^\star)$. Two scenarios present themselves. **(1)** There exists a dishonest user with attribute values $\boldsymbol{\mathfrak{a}'} = (\mathfrak{a}'_1, ..., \mathfrak{a}'_N)$ s.t. $\forall \mathfrak{a}_i \neq \varnothing$, we either have $\mathfrak{a}_i = \mathfrak{a}'_i$, or have $\mathfrak{a}_i \neq \mathfrak{a}'_i$ but issuer I_i is dishonest. **(2)** $\forall i : \mathfrak{a}_i \neq \varnothing$, issuers I_i and users holding attribute values $\mathfrak{a}_i$ are honest.

In case of scenario **(1)**, the output of $\mathcal{F}'$ is defined by $\mathcal{S}$, which instructs $\mathcal{F}'$ to return $(\mathsf{Verified}, \mathtt{false})$. In case of scenario **(2)**, $\mathcal{F}'$ outputs $(\mathsf{Verified}, \mathtt{true})$. However, thanks to the correctness of protocol Π this scenario never occurs. That is, given a signature of knowledge ϕ that was computed honestly, $\mathsf{SoK.Verify}$ will never output $\mathtt{false}$. Thus, the output of $\mathcal{F}'$ matches that of Π.

If $\mathsf{bool} = \mathtt{true}$, then thanks to the extractability property of the signature of knowledge SoK, $\mathcal{S}$ extracts witness $\mathbb{w} = (\tau, usk, upk, \psi, \sigma)$ and then executes the following steps:

- If there is no recorded entry $\langle \star, \tau, upk, \star \rangle$, then $\mathcal{S}$ aborts.
- Else if there exists $1 \leq i \leq N$ such that $\mathfrak{a}_i \neq \varnothing$ and I_i is honest, and there is no recorded entry $\langle \star, \tau, \mathfrak{a}_i, \star \rangle$, then $\mathcal{S}$ aborts.
- Otherwise, $\mathcal{S}$ sends to $\mathcal{F}'$ signature request $(\mathsf{Sign}, \boldsymbol{\mathfrak{a}}, \sigma, \mu)$.
  - If $\mathcal{F}'$ aborts, then $\mathcal{S}$ sends request $(\mathsf{Verify}, V, \boldsymbol{\mathfrak{a}}, \phi)$ to $\mathcal{F}'$.
  - Otherwise, $\mathcal{F}'$ returns $(\mathsf{PresentEND}, \phi^\star)$, and $\mathcal{S}$ sends request $(\mathsf{Verify}, V, \boldsymbol{\mathfrak{a}}, \phi^\star)$ to $\mathcal{F}'$.

  As a result, $\mathcal{S}$ receives $(\mathsf{Verified}, \mathsf{bool})$ and outputs bool in turn.

In what follows, we show that if $\mathcal{S}$ does not abort, then $\mathsf{bool} = \mathtt{true}$.

We call the two abort events Abort_1 and Abort_2. If Abort_1 does not occur, then this implies that a registration request for $\mathcal{U}$ was successfully processed by $\mathcal{S}$ and therefore $\mathcal{U}$ was successfully registered by $\mathcal{F}$. If Abort_2 does not occur then this implies one of two things: **(1)** $\forall i : \mathfrak{a}_i \neq \varnothing$, there exists an entry $\langle \star, \tau, \mathfrak{a}_i, \star \rangle$; or **(2)** $\exists j : \mathfrak{a}_j \neq \varnothing$ such that there is no recorded entry $\langle \star, \tau, \mathfrak{a}_j, \star \rangle$ but issuer I_j is dishonest. **(1)** implies that $\mathcal{S}$ issued credentials for $\mathcal{U}$ for the relevant attributes, and so did $\mathcal{F}'$. Therefore, during the $\mathsf{Present}$ call, $\mathcal{F}'$ will not abort and will return $(\mathsf{PresentEND}, \phi^\star)$, and the subsequent call $(\mathsf{Verify}, V, \boldsymbol{\mathfrak{a}}, \phi^\star)$ will result in $\mathcal{F}'$ returning $(\mathsf{Verified}, \mathtt{true})$. This matches the output of the verification in the real world. **(2)** implies that $\mathcal{U}$ is in possession of a credential that was issued by a dishonest issuer and never recorded by either $\mathcal{S}$ or $\mathcal{F}'$. In this case, $\mathcal{F}$, aborts during $\mathsf{Present}$, and when $\mathcal{S}$ calls $\mathcal{F}'$ with $(\mathsf{Verify}, V, \boldsymbol{\mathfrak{a}}, \phi)$, the output of $\mathcal{F}'$ is dictated by $\mathcal{S}$. $\mathcal{S}$ sets this output to $\mathtt{true}$ to match the output of the verification in the real world.

Therefore, this game is indistinguishable from Game 6 as long as Abort_1 and Abort_2 do not occur.

Now we show that the probability that one of these two events occur is negligible.

Abort_1 occurs if there is no recorded entry $\langle \star, \tau, upk, \star \rangle$ in the system. Again due to extraction of the verifying signature of knowledge ϕ, we can extract witness $\mathbb{w} = (\tau, usk, upk, \psi, \sigma)$. This means that $\mathcal{U}$ has knowledge of a valid ψ such that $1 \leftarrow \mathsf{DS.Verify}(pk, upk \| \tau, \psi)$ with no corresponding entry, contradicting the existential unforgeability of the digital signature scheme DS.

Abort_2 occurs if $\exists j \in [1 \,..\, N]$ such that $\mathfrak{a}_j \neq \varnothing$ and I_j is honest, and there is no recorded entry $\langle \star, \tau, \mathfrak{a}_j, \star \rangle$. Again, due to extraction of the correctly verifying signature of knowledge ϕ, we can extract a valid witness $\mathbb{w} = (\tau, usk, upk, \psi, \sigma)$. This means that σ satisfies $\mathsf{ATS.VerifyAgg}((ipk_i)_{i \in \mathbb{I}}, \tau, (\mathfrak{a}_i)_{i \in \mathbb{I}}, \sigma)$, where $\mathbb{I}$ identifies the indices s.t. $\mathfrak{a}_i \neq \varnothing$ and $j \in \mathbb{I}$. Given that there is no recorded entry $\langle \star, \tau, \mathfrak{a}_j, \star \rangle$, this implies that the honest issuer I_j has never issued a signature σ_j for pair $(\tau, \mathfrak{a}_j)$, which contradicts the existentially unforgeability under tag-based aggregation of ATS.

Game 8. This is the ideal world. This game is indistinguishable from Game 7 as functionality $\mathcal{F}'$ is identical to ideal functionality $\mathcal{F}$.

B Proof of Theorem 2

Proof. Let $\mathsf{PK} = \{pk_1, ..., pk_N\}$ be the set of challenge public keys from the existential unforgeability under tag-based aggregation experiment. Assume there exists an adversary $\mathcal{A}$ that succeeds in the unforgeability experiment by producing an aggregate signature $(\widehat{A}, B)$ on a set of messages $(m_i^*)_{i \in \mathbb{I} \subseteq [1..N]}$ under a tag τ^* such that the following holds:

$$e(B, \widehat{Q}) = e(\mathbf{G}(\tau^*), \widehat{A}) \cdot \prod_{i \in \mathbb{I}} e(\mathbf{H}(m_i^*), pk_i).$$

In addition, there exists an index $j \in \mathbb{I}$ such that the adversary does not control the secret key corresponding to pk_j, and a tag-based aggregate signature has not been issued by the signature oracle on triple (pk_j, τ^*, m_j^*). We distinguish between two types of forgeries.

Type 1 $\mathcal{A}$ has never issued a signature request for $(pk_j, \star, m_j^*)$.
Type 2 $\mathcal{A}$ has issued signature requests for (pk_j, τ, m_j^*) but $\forall \tau : \tau \neq \tau^*$.

In the following, we show that: (1) if adversary $\mathcal{A}$ outputs a Type 1 forgery, then there exists an adversary $\mathcal{B}$ that leverages $\mathcal{A}$ to break BCDH, and (2) if $\mathcal{A}$ outputs a Type 2 forgery, then there exists an adversary $\mathcal{B}$ that leverages $\mathcal{A}$ to break m-BCDH.

Type 1 Forgery. Let $\mathcal{B}$ be an adversary whose goal is to break BCDH, and hence, is given tuple $(P, X, Y, Z, \widehat{Q}, \widehat{X}, \widehat{Y}, \widehat{Z}) = (P, P^x, P^y, P^z, \widehat{Q}, \widehat{Q}^x, \widehat{Q}^y, \widehat{Q}^z)$. To compute P^{xy} and break BCDH, $\mathcal{B}$ leverages $\mathcal{A}$ and the random oracle as depicted below.

Random Oracle Programming. $\mathcal{B}$ initializes two lists $\mathcal{L}_{\mathbf{H}}$ and $\mathcal{L}_{\mathbf{G}}$, and then computes $\mathbf{H}$ and $\mathbf{G}$ as follows:

- $\mathbf{H}$: For each query m, check if there exists an entry $\langle m, \star, \mathbf{H}(m)\rangle \in \mathcal{L}_{\mathbf{H}}$, and if so, return $\mathbf{H}(m)$. If not, then do the following:
  - With probability $p_{\mathbf{H}}$, generate a random element $\rho \leftarrow \mathbb{Z}_p$, set $\mathbf{H}(m) = P^{\rho}$, and store entry $\langle m, \rho, \mathbf{H}(m)\rangle$ in $\mathcal{L}_{\mathbf{H}}$;
  - With probability $1 - p_{\mathbf{H}}$, set $\mathbf{H}(m) = Y$ and store entry $\langle m, \perp, \mathbf{H}(m)\rangle$ in $\mathcal{L}_{\mathbf{H}}$.
- $\mathbf{G}$: For each query τ, check if there exists an entry $\langle \tau, \star, \mathbf{G}(\tau)\rangle \in \mathcal{L}_{\mathbf{G}}$, and if so, return $\mathbf{G}(\tau)$. If not, then select a random element $\eta \leftarrow \mathbb{Z}_p$, set $\mathbf{G}(\tau) = P^{\eta}$, and store entry $\langle \tau, \eta, \mathbf{G}(\tau)\rangle$ in $\mathcal{L}_{\mathbf{G}}$.

CORRUPT Simulation. Without loss of generality, we assume that $\mathcal{A}$ corrupts all public keys except for one. $\mathcal{B}$ randomly selects $j \in [1 \,..\, N]$ and sets $pk_j = \widehat{X}$, and for all $k \neq j$ randomly selects $x_k \in \mathbb{Z}_p^*$ and has $sk_k = x_k$ and $pk_k = \widehat{X}_k = \widehat{Q}^{x_k}$. If $\mathcal{A}$ sends a corruption request for $pk_i \neq pk_j$, then $\mathcal{B}$ returns $sk_i = x_i$; otherwise, $\mathcal{B}$ aborts the experiment.

SIGN Simulation. On receiving a signature request (pk_i, τ, m), $\mathcal{B}$ proceeds as follows. If $i \neq j$, then $\mathcal{B}$ outputs $\sigma \leftarrow \mathsf{ATS.Sign}(\tau, sk, m)$. If $i = j$, then $\mathcal{B}$ checks if $\mathbf{H}(m) = Y$. If so, then $\mathcal{B}$ aborts, otherwise, $\mathcal{B}$ retrieves entry $\langle m, \rho, \mathbf{H}(m) = P^{\rho}\rangle$ and computes $\sigma = (\widehat{A}, B) = (\widehat{Q}^a, X^{\rho}\mathbf{G}(\tau)^a)$. Note that σ is a valid signature on pair (τ, m) relative to public key $pk_j = \widehat{X} = \widehat{Q}^x$; in fact, $\sigma = (\widehat{Q}^a, \mathbf{H}(m)^x\mathbf{G}(\tau)^a)$.

Forgery. After receiving $\mathcal{A}$'s forgery $(\mathbb{I}, (pk_i)_{i\in\mathbb{I}}, \tau^*, (m_i^*)_{i\in\mathbb{I}}, \sigma^*)$, $\mathcal{B}$ proceeds as follows. $\mathcal{B}$ checks if $\mathbf{H}(m_j^*) \neq Y$, and if so, aborts. If $\mathbf{H}(m_j^*) = Y$, then $\mathcal{B}$ parses σ^* as $(\widehat{A}^*, B^*)$, retrieves entry $\langle \tau^*, \eta^*, \mathbf{G}(\tau^*)\rangle$ from $\mathcal{L}_{\mathbf{G}}$, and computes $\Delta = \frac{B^*}{\prod_{i\in\mathbb{I}, i\neq j} \mathbf{H}(m_i^*)^{x_i}}$ followed by $\Gamma = \frac{e(\Delta, \widehat{Z})}{e(Z^{\eta^*}, \widehat{A}^*)}$.

We now prove that $\Gamma = e(P, \widehat{Q})^{xyz}$. Note that $\sigma^* = (\widehat{A}^*, B^*)$ verifies the following equality: $e(B^*, \widehat{Q}) = e(\mathbf{G}(\tau^*), \widehat{A}^*) \cdot \prod_{i\in\mathbb{I}} e(\mathbf{H}(m_i^*), \widehat{X}_i)$.

Therefore, $e(\Delta, \widehat{Q}) = e(\mathbf{G}(\tau^*), \widehat{A}^*) \cdot e(\mathbf{H}(m_j^*), \widehat{X}_j) = e(P^{\eta^*}, \widehat{A}^*)e(Y, \widehat{X})$. It follows that we have $e(Y, \widehat{X}) = \frac{e(\Delta, \widehat{Q})}{e(P^{\eta^*}, \widehat{A}^*)}$ and $e(Y, \widehat{X})^z = \frac{e(\Delta, \widehat{Z})}{e(Z^{\eta^*}, \widehat{A}^*)} = \Gamma$.

To summarize, $\mathcal{B}$ aborts in the following cases:

- If $\mathcal{A}$ sends a corruption request for $pk_i = pk_j$. As we assume that $\mathcal{A}$ corrupts all public keys except one, and we have j chosen uniformly at random, the probability that $\mathcal{B}$ *succeeds* here is $1/N$.
- If, for the forged message under the uncorrupted key pk_j, we have the case that $\mathbf{H}(m_j^*) \neq Y$, then $\mathcal{B}$ cannot extract $e(P, \widehat{Q})^{xyz}$ as required. This occurs with probability $p_{\mathbf{H}}$. Hence, the probability that $\mathcal{B}$ *succeeds* here is $1 - p_{\mathbf{H}}$.
- If any signing query (pk_j, τ, m) to the signing oracle is for a message such that $\mathbf{H}(m) = Y$, then $\mathcal{B}$ cannot produce a signature and must abort. The probability that all signing queries avoid this is $p_{\mathbf{H}}^{q_{\text{SIGN}}}$, where q_{SIGN} is the number of signing queries made with pk_j.

The probability of success is then $\frac{1}{N} \cdot (1 - p_{\mathbf{H}}) \cdot p_{\mathbf{H}}^{q_{\text{SIGN}}}$, and so the probability of aborting in the case of Type 1 forgeries is given as follows:

$$\Pr[\text{Type 1 abort}] = 1 - \left(\frac{1}{N} \cdot (1 - p_{\mathbf{H}}) \cdot p_{\mathbf{H}}^{q_{\text{SIGN}}} \right).$$

Type 2 Forgery. Let $\mathcal{B}$ be an adversary whose goal is to break m-BCDH, and therefore, has as input $(P, Y, \widehat{Q}, \widehat{X}, \widehat{Y}) = (P, P^{1/y}, \widehat{Q}, \widehat{Q}^x, \widehat{Q}^y)$. To break m-BCDH, $\mathcal{B}$ simulates the unforgeability experiment to $\mathcal{A}$ as follows.

Random Oracle Programming. $\mathcal{B}$ initializes two lists $\mathcal{L}_{\mathbf{H}}$ and $\mathcal{L}_{\mathbf{G}}$, and then computes $\mathbf{H}$ and $\mathbf{G}$ as described below:

- $\mathbf{H}$: For each query m, check if there exists an entry $\langle m, \star, \mathbf{H}(m) \rangle \in \mathcal{L}_{\mathbf{H}}$, and if so, return $\mathbf{H}(m)$. If not, then generate a random element $\rho \leftarrow \mathbb{Z}_p$, set $\mathbf{H}(m) = P^{\rho}$, and store entry $\langle m, \rho, \mathbf{H}(m) \rangle$ in $\mathcal{L}_{\mathbf{H}}$.
- $\mathbf{G}$: For each query τ, check if there exists an entry $\langle \tau, \star, \mathbf{G}(\tau) \rangle \in \mathcal{L}_{\mathbf{G}}$, and if so, return $\mathbf{G}(\tau)$. If not, then first generate a random element $\eta \in \mathbb{Z}_p$, then with probability $p_{\mathbf{G}}$, set $\mathbf{G}(\tau) = Y^{\eta}$, and with probability $1 - p_{\mathbf{G}}$, set $\mathbf{G}(\tau) = P^{\eta}$, finally, store entry $\langle \tau, \eta, \mathbf{G}(\tau) \rangle$ in $\mathcal{L}_{\mathbf{G}}$.

CORRUPT Simulation. Without loss of generality, we assume that $\mathcal{A}$ corrupts all public keys except for one. $\mathcal{B}$ randomly selects $j \in [1 \,..\, N]$ and sets $pk_j = \widehat{X}$, and for all $k \neq j$ randomly selects $x_k \in \mathbb{Z}_p^*$ and has $sk_k = x_k$ and $pk_k = \widehat{X}_k = \widehat{Q}^{x_k}$. If $\mathcal{A}$ sends a corruption request for $pk_i \neq pk_j$, then $\mathcal{B}$ returns $sk_i = x_i$; otherwise, $\mathcal{B}$ aborts the experiment.

SIGN Simulation. On receiving a signature request (pk_i, τ, m), $\mathcal{B}$ proceeds as follows. If $i \neq j$, then $\mathcal{B}$ outputs $\sigma \leftarrow \mathsf{ATS.Sign}(\tau, sk, m)$. If $i = j$, then $\mathcal{B}$ first retrieves entries $\langle m, \rho, \mathbf{H}(m) \rangle$ and $\langle \tau, \eta, \mathbf{G}(\tau) \rangle$ from $\mathcal{L}_{\mathbf{H}}$ and $\mathcal{L}_{\mathbf{G}}$ respectively. Now if $\mathbf{G}(\tau) \neq P^{\eta}$ or $\eta = 0$, then $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ randomly selects $a \in \mathbb{Z}_p$, lets $B = P^a$ and computes $\widehat{A} = (\widehat{Q}^a / \widehat{X}^{\rho})^{1/\eta}$. Note that σ is a valid signature on pair (τ, m) relative to public key $pk_j = \widehat{X} = \widehat{Q}^x$; in fact, by construction, $e(\mathbf{G}(\tau), \widehat{A}) \cdot e(\mathbf{H}(m), \widehat{X}) = e(B, P)$.

Forgery. After receiving $\mathcal{A}$'s forgery $(\mathbb{I}, (pk_i)_{i \in \mathbb{I}}, \tau^*, (m_i^*)_{i \in \mathbb{I}}, \sigma^*)$, $\mathcal{B}$ proceeds as follows. $\mathcal{B}$ first retrieves entries $\langle m^*, \rho^*, \mathbf{H}(m^*) = P^{\rho^*} \rangle$ and $\langle \tau^*, \eta^*, \mathbf{G}(\tau^*) \rangle$ from $\mathcal{L}_{\mathbf{H}}$ and $\mathcal{L}_{\mathbf{G}}$ respectively.

$\mathcal{B}$ next checks if $\mathbf{G}(\tau^*) \neq Y^{\eta^*}$ or $\rho^* = 0$, and if so, aborts. Otherwise, $\mathcal{B}$ parses σ^* as $(\widehat{A}^*, B^*)$, and computes $\Delta = \frac{B^*}{\prod_{i \in \mathbb{I}, i \neq j} \mathbf{H}(m_i^*)^{x_i}}$ followed by $\Gamma = \left(\frac{e(\Delta, \widehat{Y})}{e(P^{\eta}, \widehat{A}^*)} \right)^{1/\rho^*}$.

We show now that $\Gamma = e(P, \widehat{Q})^{xy}$. Recall that $\sigma^* = (\widehat{A}^*, B^*)$ verifies the following equality: $e(B^*, \widehat{Q}) = e(\mathbf{G}(\tau^*), \widehat{A}^*) \cdot \prod_{i \in \mathbb{I}} e(\mathbf{H}(m_i^*), \widehat{X}_i)$. Therefore: $e(\Delta, \widehat{Q}) = e(\mathbf{G}(\tau^*), \widehat{A}^*) \cdot e(\mathbf{H}(m_j^*), \widehat{X}_j) = e(Y^{\eta^*}, \widehat{A}^*) \cdot e(P^{\rho^*}, \widehat{X})$.

Hence, $e(P^{\rho^*}, \widehat{X})^y = \frac{e(\Delta, \widehat{Y})}{e(P^{\eta^*}, \widehat{A}^*)}$. This implies that $e(P, \widehat{X})^y = e(P, \widehat{Q})^{xy} = \left(\frac{e(\Delta, \widehat{Y})}{e(P^{\eta^*}, \widehat{A}^*)}\right)^{1/\rho^*}$, which equals Γ.

In the case of Type 2 forgeries, $\mathcal{B}$ aborts in the following cases:

- If $\mathcal{A}$ sends a corruption request for $pk_i = pk_j$. As we again assume $\mathcal{A}$ corrupts all public keys except one, and j is chosen uniformly at random, the probability that $\mathcal{B}$ *succeeds* here is again $1/N$.
- If, for the forged tag τ^*, we have that $\mathbf{G}(\tau^*) \neq Y^{\eta^*}$, then $\mathcal{B}$ cannot extract $e(P, \widehat{Q})^{xy}$ as required. This occurs with probability $1 - p_{\mathbf{G}}$. Hence, the probability that $\mathcal{B}$ *succeeds* here is $p_{\mathbf{G}}$.
- If any signing query (pk_j, τ, m) to the signing oracle is for a tag such that $\mathbf{G}(\tau) \neq Y^{\eta}$, then $\mathcal{B}$ cannot produce a signature and must abort. The probability that all signing queries avoid this is $p_{\mathbf{G}}^{q_{\text{SIGN}}}$, where q_{SIGN} is the number of signing queries made with pk_j.

The probability of success is then $\frac{1}{N} \cdot p_{\mathbf{G}}^{q_{\text{SIGN}}+1}$, and so the probability of aborting in the case of Type 2 forgeries is given as follows:

$$\Pr[\text{Type 2 abort}] = 1 - \left(\frac{1}{N} \cdot p_{\mathbf{G}}^{q_{\text{SIGN}}+1}\right).$$

C Instantiation of the SoK

Let $\mathcal{U}$ be a user equipped with (1) a pair of secret and public key $(usk, upk) = (usk, P^{usk})$, (2) an AGHO signature $\psi = (\widehat{R}, \widehat{S}, T)$ on pair $(upk, \Gamma = \mathbf{G}(\tau))$ from registration authority RA, and (3) a tag-based aggregate signature $\sigma = (\widehat{A}, B)$ on attributes $\{\mathfrak{a}_i\}_{i \in \mathbb{I}}$ that verifies against the public keys of issuers I_i with $i \in \mathbb{I}$. Upon receiving a nonce μ from some verifier $\mathcal{V}$, $\mathcal{U}$ presents their attribute values $\{\mathfrak{a}_i\}_{i \in \mathbb{I}}$ by executing the following steps.

$\mathcal{U}$ randomizes signatures $\sigma' \leftarrow \mathsf{ATS.Randomize}(\sigma)$ and $\psi' \leftarrow \mathsf{AGHO.Randomize}(\psi)$. We parse σ' as $(\widehat{A}', B')$ and ψ' as $(\widehat{R}', \widehat{S}', T')$. Next, $\mathcal{U}$ samples $\delta \in \mathbb{Z}_p^*$ and computes the following:

$$\forall i \in \mathbb{I}: \; H_i = \mathbf{H}(\mathfrak{a}_i)^\delta \; ; \quad \bar{P} = P^\delta; \quad \bar{T} = T'^\delta; \quad \bar{B} = B'^\delta; \quad \bar{\Gamma} = \Gamma^\delta;$$
$$\bar{V} = V^\delta; \quad \bar{upk} = upk^\delta.$$

Let $pp = P$; $\mathbb{x} = (\bar{upk}, \{\mathfrak{a}_i, H_i\}_{i \in \mathbb{I}}, \bar{P})$; $\mathbb{w} = (\delta, usk)$, and let $\mathcal{R}$ be the relation defined as $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$, if and only if, the following holds:

$$\bar{P} = P^\delta \; ; \; \bar{V} = V^\delta \; ; \; H_i = \mathbf{H}(\mathfrak{a}_i)^\delta \; ; \; \bar{upk} = \bar{P}^{usk} \tag{3}$$

$\mathcal{U}$ now computes η, a SoK for $\mathcal{R}$, and outputs presentation

$$\phi = (\eta, \bar{P}, \bar{V}, \widehat{R}', \widehat{S}', \bar{T}, \widehat{A}', \bar{B}, \bar{\Gamma}, \bar{upk}, \{\mathfrak{a}_i, H_i\}_{i \in \mathbb{I}}).$$

$\mathcal{V}$ accepts ϕ if $\bar{P} \neq 1$, η is a valid SoK for the relation depicted in equation 2 and the following equalities hold:

$$e(\bar{P}, \widehat{Q}) = e(\bar{T}, \widehat{R}') \cdot e(\bar{upk}, \widehat{U}_1) \cdot e(\bar{\Gamma}, \widehat{U}_2) \ ; \ e(\bar{V}, \widehat{R}') = e(\bar{P}, \widehat{S}') \ ; \tag{4}$$

$$e(\bar{B}, \widehat{Q}) = \prod_{i \in \mathbb{I}} e(H_i, \widehat{X}_i) \cdot e(\bar{\Gamma}, \widehat{A}'). \tag{5}$$

Next, we describe first how η is computed. This is followed by showing that if $\mathcal{U}$ is able to produce a valid η, then we can extract a valid witness for the relation defined in Equation 1. Finally, we prove that under the DDH assumption in $\mathbb{G}_1$, ϕ is simulatable, i.e., it does not leak any information whatsoever about the witness.

C.1 η Computation

We first describe the interactive zero-knowledge proof underlying the computation of η, and then show how to apply the Fiat-Shamir heuristics to the proof to finally obtain η.

User $\mathcal{U}$ proves knowledge of a valid witness $\mathbb{w} = (\delta, usk)$ for the relation depicted in equation 3, by executing the following:

- Select two random numbers α and β in $\mathbb{Z}_p$;
- Compute and send tuple $(A, B, \{C_i\}_{i \in \mathbb{I}}, D) = (P^\alpha, V^\alpha, \{\mathbf{H}(\mathfrak{a}_i)^\alpha\}_{i \in \mathbb{I}}, \bar{P}^\beta)$ to the verifier;
- On receiving challenge c from the verifier, compute and send $\pi_1 = \alpha + c \cdot \delta$ and $\pi_2 = \beta + c \cdot usk$.

The verifier accepts π_1 and π_2 if the following equalities hold; $P^{\pi_1} = A \cdot \bar{P}^c$; $V^{\pi_1} = B \cdot \bar{V}^c$; $\mathbf{H}(\mathfrak{a}_i)^{\pi_1} = C_i \cdot H_i^c$; $\bar{P}^{\pi_2} = D \cdot \bar{upk}^c$.

To produce η on input of a message μ, we apply the Fiat-Shamir heuristics to generate the challenge c. Let $\mathcal{H}_p : \{0,1\}^* \rightarrow \mathbb{Z}_p$ be a cryptographic hash function. $\mathcal{U}$ computes η by first computing tuple $(A, B, \{C_i\}_{i \in \mathbb{I}}, D)$, followed by $c = \mathcal{H}_p(A, B, \{C_i\}_{i \in \mathbb{I}}, D, \mu)$, and outputting $\eta = (A, B, \{C_i\}_{i \in \mathbb{I}}, D, \pi_1, \pi_2)$.

C.2 Extractability

We recall that $\phi = (\eta, \bar{P}, \bar{V}, \widehat{R}', \widehat{S}', \bar{T}, \widehat{A}', \bar{B}, \bar{\Gamma}, \bar{upk}, \{\mathfrak{a}_i, H_i\}_{i \in \mathbb{I}})$.

Note that given the extractability property of signatures of knowledge, if a user is able to produce a valid signature η for relation $\mathcal{R}$, then it is possible to extract the corresponding witness (δ, usk) that verifies $\bar{P} = P^\delta$; $\bar{V} = V^\delta$; $H_i = \mathbf{H}(\mathfrak{a}_i)^\delta$; $\bar{upk} = \bar{P}^{usk}$. Now given that $\bar{P} \neq 1$, then $\delta \neq 0$ and we can write $P = \bar{P}^{1/\delta}$, $V = \bar{V}^{1/\delta}$ and $\bar{upk}^{1/\delta} = \bar{P}^{usk/\delta} = P^{usk}$. Furthermore, given the equalities in equation 4, we can deduce that:

$$e(\bar{V}^{1/\delta}, \widehat{R}) = e(V, \widehat{R}) = e(\bar{P}^{1/\delta}, \widehat{S}) = e(P, \widehat{S}) \ ;$$

$$e(\bar{T}^{1/\delta}, \widehat{R}) \cdot e(\bar{upk}^{1/\delta}, \widehat{U}_1) \cdot e(\bar{\Gamma}^{1/\delta}, \widehat{U}_2) = e(\bar{P}^{1/\delta}, \widehat{Q}) = e(P, \widehat{Q}) \ ;$$

$$e(\bar{B}^{1/\delta}, \widehat{Q}) = e(\bar{\Gamma}^{1/\delta}, \widehat{A}) \cdot \prod_{i \in \mathbb{I}} e(H_i^{1/\delta}, \widehat{X}_i).$$

That is, $(\widehat{A}, \bar{B}^{1/\delta})$ is a valid tag-based aggregate signature on the pre-image of $\bar{\Gamma}^{1/\delta}$ and attributes $\mathfrak{a}_i$, and that $(\widehat{R}, \widehat{S}, \bar{T}^{1/\delta})$ is a valid AGHO signature on pair $\bar{upk}^{1/\delta}, \bar{\Gamma}^{1/\delta}$.

To extract the pre-image of $\bar{\Gamma}^{1/\delta}$, we leverage the following:

1. Random oracle $\mathbf{G}$, which is initialized with two maps L_1 and L_2, and which for each query τ, checks if $L_1[\tau]$ is empty. If so, it selects a random group element $\Gamma \in \mathbb{G}_1$, and sets $L_1[\tau] \leftarrow \Gamma$ and $L_2[\Gamma] \leftarrow \tau$. Finally, returns $L_1[\tau]$.
2. The fact that registration authority RA only signs well-formed pairs defined as $(upk, \mathbf{G}(\tau))$.

Notice that if $(\widehat{R}, \widehat{S}, \bar{T}^{1/\delta})$ is a valid AGHO signature on pair $(\bar{upk}^{1/\delta}, \bar{\Gamma}^{1/\delta})$, then under the existential unforgeability of AGHO signatures, this signature corresponds to either a signature or a randomization of a signature computed by RA on the same pair. Since RA is trusted to sign only well-formed pairs, then this entails that $\bar{\Gamma}^{1/\delta}$ was computed by calling $\mathbf{G}$ on some input $\tau \in \{0,1\}^*$, and that τ corresponds to $L_2[\bar{\Gamma}^{1/\delta}]$.

Hence the extractability property of signatures of knowledge guarantees that if user $\mathcal{U}$ is able to produce a valid SoK η, then we can extract a valid witness (δ, usk), and that $(\widehat{A}, \bar{B}^{1/\delta})$ is a valid tag-based aggregate signature on the pre-image of $\bar{\Gamma}^{1/\delta}$ and attributes $\mathfrak{a}_i$, and that $(\widehat{R}, \widehat{S}, \bar{T}^{1/\delta})$ is a valid AGHO signature on pair $\bar{upk}^{1/\delta}, \bar{\Gamma}^{1/\delta}$.

C.3 Simulatability

We now show that if DDH holds in $\mathbb{G}_1$, we can simulate a valid presentation ϕ without access to a valid witness.

- Randomly select $\alpha, \beta, \gamma, \delta, \nu, \{\eta_i\}_{i\in\mathbb{I}}, \theta \in \mathbb{Z}_p^*$;
- Compute $\bar{P} = P^\alpha$, $\bar{V} = P^\beta$, $\bar{T} = P^\gamma$, $\bar{B} = P^\delta$, $\bar{\Gamma} = P^\nu$, $H_i = P^{\eta_i}$, and $\bar{upk} = P^\theta$ (note that if the DDH assumption holds in $\mathbb{G}_1$, then it is computationally infeasible to tell whether tuple $(\bar{P}, \bar{V}, \bar{T}, \bar{B}, \bar{\Gamma})$ was generated following the description in Section 5.3 or as random group elements);
- Compute $\widehat{R}' = (\widehat{Q}^\alpha \widehat{U}_1^{-\theta} \widehat{U}_2^{-\nu})^{1/\gamma}$, $\widehat{S}' = \widehat{R}'^{\beta/\alpha}$, and $\widehat{A}' = (\widehat{Q} \prod_{i\in\mathbb{I}} \hat{ipk}_i^{-\eta_i})^{\delta/\nu}$ (by construction, tuple $(\bar{P}, \bar{V}, \widehat{R}', \widehat{S}', \bar{T}, \widehat{A}', \bar{B}, \bar{\Gamma}, \bar{upk}, \{\mathfrak{a}_i, H_i\}_{i\in\mathbb{I}})$ verifies equations 4);
- Pick π_1, π_2 and c randomly in $\mathbb{Z}_p$;
- Compute $A = P^{\pi_1} \cdot \bar{P}^{-c}$; $B = V^{\pi_1} \cdot \bar{V}^{-c}$; $C_i = \mathbf{H}(\mathfrak{a}_i)^{\pi_1} \cdot H_i^{-c}$; $D = \bar{P}^{\pi_2} \cdot \bar{upk}^{-c}$;
- Using random oracle $\mathcal{H}_p$, define the hash $\mathcal{H}_p(A, B, \{C_i\}_{i\in\mathbb{I}}, D, \mu)$ as value c;
- Finally, set η to tuple $(A, B, \{C_i\}_{i\in\mathbb{I}}, D, \pi_1, \pi_2)$.