Efficient Aggregate Anonymous Credentials for Decentralized Identity

Rebekah Mercer, Kaoutar Elkhiyaoui, Angelo De Caro, and Elli Androulaki

IBM Research – Zurich
rebekah.mercer@ibm.com, {kao,adc,lli}@zurich.ibm.com

Anonymous credential schemes allow users to prove claims about themselves in a privacy-preserving manner. Put simply, a user can prove that she has an attribute value (e.g., she is a citizen) without revealing any other information about herself. Traditional schemes (including those with multiple credential issuers) operate under the assumption that each recognized issuer produces credentials for all user attributes. This assumption, however, is not practical: in the real world, no such issuer exists; an identity provider today issues a user credentials for a subset of user attributes, not all. A promising approach to support this setting is *aggregate anonymous credentials*, which allow a user to receive credentials from several issuers such that she can aggregate them and prove various claims about her attribute values in one shot. In this paper, we first introduce what we call *aggregate tag-based signatures* and describe an efficient instantiation. We then leverage the latter together with structure-preserving signatures and signatures of knowledge to construct an efficient aggregate anonymous credential scheme. We finally, formally evaluate the security of the proposed schemes and run benchmarks to showcase the practicality of the resulting scheme and its relevance for decentralized identity applications.

1 Introduction

Anonymous credentials were introduced to enable authentication while maintaining user privacy. In traditional systems, proving attributes such as age, group membership, or citizenship often requires disclosing additional sensitive information. Anonymous credential systems address this by enabling users to selectively disclose certified attributes while preserving unlinkability and non-transferability. Foundational work by Chaum [15] and later by Camenisch and Lysyanskaya [9] established privacy-preserving authentication protocols that give users control over their personal information. However, designing secure, efficient, and scalable anonymous credentials, especially in decentralized settings, remains a challenge.

In decentralized identity systems, multiple issuers may independently certify different attribute values for the same user. Existing anonymous credential schemes, however, often assume that an issuer (which can be potentially distributed) certifies all user attribute values. In this paper, we address this limitation by introducing an aggregate anonymous credentials scheme, which allows credentials received from different issuers to be combined into a single and compact credential that can be used to *efficiently* prove claims over user attribute

values. We consider a model where (1) each attribute is independently certified by one of the issuers, bringing us closer to real-world scenarios–where, for instance, dedicated government agencies are each tasked with issuing a specific type of credential (e.g., driver's licenses or residency permits), and where a *trusted registration authority* (RA) registers users' public keys. A user starts her journey in the system by registering with RA, which binds the user's public key to a *unique tag*, by producing a signature over the pair (`public key`, `tag`), which later serves as a proof of registration. When the now-registered user seeks to obtain an attribute credential from one of the issuers, she presents the RA's signature and proves knowledge of the corresponding secret key. The issuer then uses what we call *tag-based aggregate signatures* to sign the user's attribute value together with the user's tag. The use of these signatures achieves two goals: (1) only attribute credentials associated with the same tag can be aggregated, which prevents malicious users and issuers from combining credentials across identities and forging new ones; (2) users can aggregate their attribute credentials efficiently and non-interactively (i.e., without requiring coordination between issuers). Finally, the user demonstrates possession of certain attribute values while preserving her anonymity by first aggregating the relevant attribute credentials and proving the following statements in zero-knowledge: (1) she knows the secret key of a registered public key; (2) she holds a valid aggregate credential that certifies the disclosed attribute values; (3) the tag associated with the registered public key is the same as the tag associated with the aggregated credentials. At no point during this operation is the tag or the public key revealed, protecting hence, user privacy.

Contributions. The contributions of this paper are as follows.

- We formally define tag-based aggregate signatures and propose an aggregate anonymous credentials scheme that leverages, in a black-box manner, digital signatures, tag-based aggregate signatures and signatures of knowledge.
- We prove the security of the resulting black-box construction using the real/ideal world paradigm.
- We introduce an efficient instantiation of tag-based aggregate signatures and analyze its security in the random oracle model.
- We finally instantiate the black-box construction using AGHO structure preserving signatures [2], our tag-based aggregate signature, and Schnorr proofs, and evaluate its performances.

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 the following participants:

The **registration authority** (RA), 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 public key is unique and verifying that a user advertizing a given public key actually holds the corresponding secret key.

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. This implies that each type of attribute is uniquely associated with an issuer.

A set of **users** such that each user $\mathcal{U}$ is equipped with a pair of secret and public keys (usk, upk), and every now and then, is required to *anonymously* prove to a verifier $\mathcal{V}$ that she possesses values $\mathfrak{a}_1, ..., \mathfrak{a}_l$ of a set of attributes $\mathtt{att}_1, ..., \mathtt{att}_l$. A user $\mathcal{U}$ will first register in the system, by interacting with registration authority RA. A user who would like to certify the value of a given attribute will contact the relevant issuer I, who will first verify the provided value and then return the valid credential, which can then be *aggregated* with the other valid credentials that $\mathcal{U}$ possess, which $\mathcal{U}$ will use to prove statements on values $\mathfrak{a}_1, ..., \mathfrak{a}_l$ to $\mathcal{V}$.

A set of **verifiers** $\{\mathcal{V}_1, \mathcal{V}_2, \dots\}$ 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 $\mathcal{U}$ *proving predicates* on 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 (i.e., prove a predicate on the value of her attributes) if and only if she received credentials from the authorized issuers that certify her attribute values. Unlinkability assures users that presentations are *indistinguishable* between users holding the same attribute values.

Furthermore, anonymous aggregate credentials have the added feature of **aggregatability**: A user receives a credential for each of her attribute values from the relevant issuer, and at time of credentials presentation, she aggregates her credentials and *efficiently* prove predicates over multiple attribute values.

Trust Model. Our trust assumptions are as follows: The **registration authority** is honest-but-curious: it follows the protocol but may collude with other system participants to de-anonymize the presentations of honest users. **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.

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 that to behave *arbitrarily*.

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.

Ideal Functionality $\mathcal{F}$ (cf. Figure 1). Ideal functionality $\mathcal{F}$ represents a trusted third party, which executes the operations of 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 produce the corresponding (correct) outputs, which, in turn, are transmitted to the parties authorized to access them. In more details, $\mathcal{S}$ and *honest* system participants submit their inputs to $\mathcal{F}$ through interfaces Register, Issue, Aggregate, Sign and Verify.

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

Issue is invoked by a user $\mathcal{U}$ to certify one of her attribute values $\mathfrak{a}$ by the relevant authorized issuer I. This results in $\mathcal{U}$ obtaining a credential to use in subsequent presentations.

Aggregate is invoked by a user $\mathcal{U}$ to aggregate her credentials for some attribute values $\vec{\mathfrak{a}}$ in a way that enables her to successfully present them later. Aggregation is a local operation, and hence, $\mathcal{S}$ does not learn any information during an Aggregate call.

Sign is queried by a user $\mathcal{U}$ to present some attribute values $\vec{\mathfrak{a}}$ while signing a message m. If $\mathcal{U}$ has previously succeeded in calling Aggregate with $\vec{\mathfrak{a}}$, then $\mathcal{F}$ accepts the query; otherwise, it validates it against Cred. This captures the fact that even if $\mathcal{U}$ does not invoke Aggregate, $\mathcal{F}$ will still accept the query, as long as $\mathcal{U}$ holds the relevant credentials. If $\mathcal{F}$ accepts, it randomly generates a signature ζ. The randomness of ζ ensures that anyone receiving triple $(\vec{\mathfrak{a}}, m, \zeta)$ (including $\mathcal{S}$) will not learn any information about $\mathcal{U}$ beyond what's revealed by $\vec{\mathfrak{a}}$.

Verify is invoked by a verifier $\mathcal{V}$ to check the validity of credential presentation $(\vec{\mathfrak{a}}, m, \zeta)$. If the presentation has already been produced by a valid Sign call,

Register. On input 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 $\mathsf{Cred}[\mathcal{U}][i]$ denote the ith element in the array).
4. Return 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 a credential to each user only once.)*
   (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 $(\mathsf{Aggregate}, \vec{\mathfrak{a}})$ from $\mathcal{U}$, where $\vec{\mathfrak{a}} = (\mathfrak{a}_1, ..., \mathfrak{a}_N)$ 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 $\mathsf{AggCred}[\sigma] \leftarrow \langle \mathcal{U}, \vec{\mathfrak{a}} \rangle$.
   (c) Return $(\mathsf{AggregateEND}, \sigma)$ to $\mathcal{U}$. *(Users who hold non-empty attribute values successfully aggregate the corresponding credentials.)*
3. Else abort.

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

1. If $\mathsf{AggCred}[\sigma] \rightarrow \langle \mathcal{U}, \vec{\mathfrak{a}} \rangle$, then:
   (a) Randomly generate ζ.
   (b) Set $\mathsf{Signed}[\vec{\mathfrak{a}}, m, \zeta] \leftarrow \mathsf{true}$.
   (c) Return $(\mathsf{SignEND}, \zeta)$ to $\mathcal{U}$ and $(\mathsf{SignEND}, \vec{\mathfrak{a}}, m, \zeta)$ 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 ζ.
   (b) Set $\mathsf{Signed}[\vec{\mathfrak{a}}, m, \zeta] \leftarrow \mathsf{true}$.
   (c) Return $(\mathsf{SignEND}, \zeta)$ to $\mathcal{U}$ and $(\mathsf{SignEND}, \vec{\mathfrak{a}}, m, \zeta)$ to $\mathcal{S}$. *(Users with disclosed attribute values successfully sign messages even if they did not call* Aggregate *before.)*
4. Else abort.

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

1. If $\mathsf{Signed}[\vec{\mathfrak{a}}, m, \zeta] \rightarrow \mathsf{false}$, then abort.
2. Else if $\mathsf{Signed}[\vec{\mathfrak{a}}, m, \zeta] \rightarrow \mathsf{true}$, then:
   (a) Return $(\mathsf{Verified}, \mathtt{true})$ to $\mathcal{V}$. *(Completeness.)*
   (b) Send $(\mathsf{Verified}, \mathcal{V}, \vec{\mathfrak{a}}, m, \zeta, \mathtt{true})$ to $\mathcal{S}$.
3. Else if $\mathsf{Signed}[\vec{\mathfrak{a}}, m, \zeta] \rightarrow \mathsf{nil}$, then:
   (a) If there exists a dishonest user $\mathcal{U}'$ with attribute values $\vec{\mathfrak{a}'} = (\mathfrak{a}'_1, ..., \mathfrak{a}'_N)$ 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, then query $\mathcal{S}$ with $(\mathsf{Verify}, \vec{\mathfrak{a}}, m, \zeta)$. On receiving the response $(\mathsf{Verified}, \mathsf{bool})$, do:
      i. Set $\mathsf{Signed}[\vec{\mathfrak{a}}, m, \zeta] \leftarrow \mathsf{bool}$.
      ii. Return $(\mathsf{Verified}, \mathsf{bool})$ to $\mathcal{V}$.
   (b) Else:
      i. Set $\mathsf{Signed}[\vec{\mathfrak{a}}, m, \zeta] \leftarrow \mathsf{false}$.
      ii. Return $(\mathsf{Verified}, \mathtt{false})$ to $\mathcal{V}$. *(Unforgeability.)*
      iii. Send $(\mathsf{Verified}, \mathcal{V}, \vec{\mathfrak{a}}, m, \zeta, \mathtt{false})$ to $\mathcal{S}$.

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

then $\mathcal{F}$ accepts and returns `true`. If the presentation is marked as invalid, then $\mathcal{F}$ rejects and returns `false`. If the presentation has not been seen before, then $\mathcal{F}$ checks whether the presentation could have been created by corrupt users who either hold the disclosed attributes in $\vec{\mathtt{a}}$ or collude with issuers in charge of producing credentials for those disclosed attributes. If that's the case, then $\mathcal{F}$ refers to $\mathcal{S}$ and returns what $\mathcal{S}$ returns. If not, then $\mathcal{S}$ records the presentation, labels it as invalid, and outputs `false`.

We say that a protocol Π *securely realizes* ideal functionality $\mathcal{F}$ in the standalone model if, for every PPT adversary $\mathcal{A}$ against Π, there exists a PPT simulator $\mathcal{S}$ interacting with $\mathcal{F}$, such that: the distribution $\textsc{Exec}_{\Pi,\mathcal{A}}(1^\kappa)$ of the outputs of the honest parties and $\mathcal{A}$, during an execution of Π in the real world, is computationally indistinguishable from the distribution of $\textsc{Exec}_{\mathcal{F},\mathcal{S}}(1^\kappa)$ of the outputs of the honest parties and $\mathcal{S}$, interacting with $\mathcal{F}$ in the ideal world. That is: $\textsc{Exec}_{\Pi,\mathcal{A}}(1^\kappa) \approx_c \textsc{Exec}_{\mathcal{F},\mathcal{S}}(1^\kappa)$, with κ the security parameter, and $\approx_c$ denoting 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 1 if ψ is a valid signature of m relative to pk, and 0 otherwise.

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 allow a prover to convince a verifier that they know a witness $\mathbb{w}$ for a public statement $\mathbb{x}$, binding the proof to a message m [14]. The following definitions are based heavily on those found in the game-based definition of SimExt-security given in [14]. Notably, we drop the variable capturing the relation as an explicit input to all algorithms, rather having the relation $\mathcal{R}$ captured inside of pp.

Definition. Let $\mathcal{R}$ be a binary relation. For pairs $(\mathbb{w}, \mathbb{x})$ with $\mathcal{R}(\mathbb{w}, \mathbb{x}) = 1$, $\mathbb{w}$ is called the witness and $\mathbb{x}$ the public input. A signature of knowledge for $\mathcal{R}$ consists of the following algorithms:

- $\mathsf{SoK.Setup}(\mathcal{R}) \to pp$: on input relation $\mathcal{R}$, output public parameters pp.

– $\mathsf{SoK.Sign}(pp, \mathbb{x}, \mathbb{w}, m) \to \zeta$: on input public parameters pp, and $\mathbb{x}, \mathbb{w}$ such that $\mathcal{R}(\mathbb{w}, \mathbb{x}) = 1$, and message m, output a signature of knowledge ζ.
– $\mathsf{SoK.Verify}(pp, \mathbb{x}, m, \zeta) \to \{0, 1\}$: on input public parameters pp, $\mathbb{x}$, message m, and signature ζ, output 1 if ζ is a valid signature of knowledge and 0 otherwise.

A signature of knowledge satisfies the properties correctness, simulatability, and extraction, as defined below. To capture simulatability and extraction, we rely on 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 $\mathsf{SoK.Setup}(\mathcal{R})$. The trapdoor td is used to both simulate signatures and to extract witnesses from adversarial signatures.
– $\mathsf{SoK.SimSign}(pp, td, \mathbb{x}, m) \to \zeta$: on input public parameters pp, trapdoor td, public input $\mathbb{x}$, and message m, output simulated signature ζ. The distribution of simulated signatures is computationally indistinguishable from that of signatures generated by the signing algorithm.
– $\mathcal{E}(pp, td, \mathbb{x}, m, \zeta) \to \hat{\mathbb{w}}$: on input public parameters pp, trapdoor td, public input $\mathbb{x}$, message m, and signature ζ, output extracted witness $\hat{\mathbb{w}}$. If ζ verifies under $(pp, \mathbb{x}, m)$, $\hat{\mathbb{w}}$ is a valid witness, with $(\mathbb{x}, \hat{\mathbb{w}}) \in \mathcal{R}$.

Correctness. For all $\mathbb{x} \in \mathcal{R}$, for all valid witnesses $\mathbb{w}$ for $\mathbb{x}$, (i.e. such that $\mathcal{R}(\mathbb{w}, \mathbb{x}) = 1$), and for all m in the message space, the following holds:

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

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:

$$\left|\Pr\left[b = 1 \;\middle|\; \begin{array}{c} (pp, td) \leftarrow \mathsf{SoK.SimSetup}(\mathcal{R}) \\ b \leftarrow \mathcal{A}^{\mathcal{S}(pp, td, \cdot, \cdot, \cdot)}(s, pp) \end{array}\right] - \Pr\left[b = 1 \;\middle|\; \begin{array}{c} pp \leftarrow \mathsf{SoK.Setup}(\mathcal{R}) \\ b \leftarrow \mathcal{A}^{\mathsf{SoK.Sign}(pp, \cdot, \cdot, \cdot)}(s, pp) \end{array}\right]\right| \leq \varepsilon(\kappa), \quad (2)$$

where the oracle $\mathcal{S}$ receives the values $(\mathbb{x}, \mathbb{w}, m)$ as inputs, checks that $\mathcal{R}(\mathbb{w}, \mathbb{x}) = 1$, and returns $\zeta \leftarrow \mathsf{SoK.SimSign}(pp, td, \mathbb{x}, m)$, with td the trapdoor needed for the simulator to generate signatures without a witness.

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 s

polynomial in κ, we have the following:

$$\Pr\left[\begin{array}{c}\mathcal{R}(\hat{\mathbb{w}},\mathbb{x})=1 \ \vee \ (\mathbb{x},m)\in\mathcal{Q}\vee\\ \mathsf{SoK.Verify}(pp,\mathbb{x},m,\zeta)=0\end{array}\ \middle|\ \begin{array}{c}(pp,td)\leftarrow\mathsf{SoK.SimSetup}(\mathcal{R})\\ (\mathbb{x},m,\zeta)\leftarrow\mathcal{A}^{\mathcal{S}(pp,td,\cdot,\cdot,\cdot)}(s,pp)\\ \hat{\mathbb{w}}\leftarrow\mathcal{E}(pp,td,\mathbb{x},m,\zeta)\end{array}\right]=1-\varepsilon(\kappa), \tag{3}$$

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

3.3 Aggregate Tag-based Signatures (ATS)

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

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

Aggregate tag-based signatures should satisfy correctness and existentially unforgeability, as defined below.

Correctness. For all $i\in[1\,..\,n]$, generate key pairs $(sk_i,pk_i)\leftarrow\mathsf{ATS.Keygen}(1^\kappa)$. Then run $\mathsf{ATS.Sign}(sk_i,\tau,m_i)$ to generate signatures σ_i. Then form aggregate signature by running $\sigma\leftarrow\mathsf{ATS.Aggregate}((\sigma_i)_{i\in[1..n]})$. We say that correctness holds if, $\forall i\in[1\,..\,n]$, the following holds:

$$\mathsf{ATS.Verify}(pk_i,\tau,m_i,\sigma_i)=1\ \wedge\ \mathsf{ATS.VerifyAgg}((pk_i)_{i\in[1..n]},\tau,(m_i)_{i\in[1..n]},\sigma)=1.$$

Existential Unforgeability under Tag-based Aggregation. Existential unforgeability under tag-based aggregation holds iff for any adversary $\mathcal{A}$ the probability that $\mathcal{A}$ wins in the following is negligible.

- Let $\mathsf{PK}=\{pk_i\}_{i\in[1..N]}$ be the set of challenge public keys.

– Let CORRUPT be an oracle which on input of public key $pk_i \in \mathsf{PK}$ returns the corresponding sk_i, and SIGN be an oracle which on input of triple (pk_i, τ_i, m_i) returns σ_i such that $1 \leftarrow \mathsf{ATS.Verify}(pk_i, \tau_i, m_i, \sigma_i)$.
– Let $\mathcal{C}$ denote the set of queries pk_i to CORRUPT, and Σ the set of queries (pk_i, τ_i, m_i) to SIGN.

1. $(\mathcal{C}, \Sigma) \leftarrow \mathcal{A}^{\text{CORRUPT,SIGN}}$;
2. $(\mathbb{I} \subset [1 \,..\, N], (pk_i)_{i\in\mathbb{I}}, \tau, (m_i)_{i\in\mathbb{I}}, \sigma) \leftarrow \mathcal{A}$.

We say that $\mathcal{A}$ wins the experiment if the following holds:

$$\mathsf{ATS.VerifyAgg}((pk_i)_{i\in\mathbb{I}}, \tau, (m_i)_{i\in\mathbb{I}}, \sigma) = 1 \;\wedge\; \exists\, i \in \mathbb{I} : pk_i \in \mathsf{PK}\backslash\mathcal{C} \;\wedge\; (pk_i, \tau, m_i) \notin \Sigma.$$

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 generates system-wide public parameters pp. RA samples a key pair $(sk, pk) \leftarrow \mathsf{DS.Keygen}(1^\kappa)$ for the digital signature scheme DS and publishes pk.

Each issuer I_i 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 $(usk, upk) \leftarrow \mathsf{DS.Keygen}(1^\kappa)$ and sends upk to RA together with a signature of knowledge proving $\mathcal{U}$ knows usk, on a fresh message m from RA. If this signature of knowledge verifies, RA assigns $\mathcal{U}$ a unique tag τ and issues $\psi \leftarrow \mathsf{DS.Sign}(sk, upk\|\tau)$.

The user initializes her credential state as

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

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

IssueCred. On input $(\tau, upk, \psi, \mathfrak{a}_i)$ from $\mathcal{U}$, issuer I_i first sets $\gamma = upk\|\tau$, and checks that $\mathsf{DS.Verify}(pk, \gamma, \psi) = 1$. If not, it rejects. Otherwise, it verifies that $\mathcal{U}$ knows usk via a signature of knowledge. If this check succeeds, the issuer computes $\sigma_i \leftarrow \mathsf{ATS.Sign}(isk_i, \tau, \mathfrak{a}_i)$, and the user sets $\mathsf{cred}[i] = (\mathfrak{a}_i, \sigma_i)$.

Sign. Let $\mathsf{CRED} = (\tau, usk, upk, \psi, \mathsf{cred}[1], \ldots, \mathsf{cred}[N])$, where $\mathsf{cred}[i] = (\mathfrak{a}_i, \sigma_i)$. To disclose attributes $(\mathfrak{a}_i)_{i\in\mathbb{I}}$ to a verifier $\mathcal{V}$, the user computes $\sigma \leftarrow \mathsf{ATS.Aggregate}(\{\sigma_i\}_{i\in\mathbb{I}})$. The user then computes a signature of knowledge $\zeta \leftarrow \mathsf{SoK.Sign}(\mathbb{w}, \mathbb{x}, m, pp)$, where $\mathbb{w} = (\tau, usk, upk, \psi, \sigma)$ and $\mathbb{x} = (pk, \{ipk_i, \mathfrak{a}_i\}_{i\in\mathbb{I}})$, proving that

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

Verify. On input $(\mathbb{x}, m, \zeta, pp)$, verifier outputs 1 iff $\mathsf{SoK.Verify}(\mathbb{x}, m, \zeta, pp) = 1$.

Theorem 1. *Let* DS *be a digital signature scheme existentially unforgeable under chosen-message attacks,* ATS *be an aggregate tag-based signature scheme existentially unforgeable under tag-based aggregation, and* SoK *satisfy zero-knowledge, knowledge-soundness, and simulation-extractability. Then the above protocol securely realizes the ideal functionality $\mathcal{F}$ in the standalone model.*

The proof of Theorem 1 is deferred to Appendix A.

5 Instantiation

In this section, we describe how to instantiate the building blocks described in Section 3. Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ be three cyclic groups of large prime order p that admit a non-degenerate bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Let $\mathbf{H} : \{0,1\}^* \to \mathbb{G}_1$ and $\mathbf{G} : \{0,1\}^* \to \mathbb{G}_1$ be two cryptographic hash functions. Let P and $\widehat{Q}$ denote the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. We denote elements in $\mathbb{G}_1$ by upper-case letters, whereas elements in $\mathbb{G}_2$ are denoted by upper-case letters with hat. Elements in $\mathbb{Z}_p$ are denoted by lower-case letters.

In our 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 implemented using AGHO structure preserving signatures [2] that sign a pair $(upk, \mathbf{G}(\tau)) \in \mathbb{G}_1 \times \mathbb{G}_1$ composed of a user's public key and the hash of that user's tag τ (cf. Section 5.1). The aggregate tag-based signature ATS is instantiated with our own construction detailed in Section 5.2. The signature of knowledge is instantiated by applying the Fiat-Shamir heuristic to a Sigma protocol as described in Appendix C.

5.1 AGHO Structure-Preserving Signature Scheme

We use the AGHO structure preserving signature (SPS) scheme [2] to instantiate DS. The AGHO scheme supports signing vectors of elements in $\mathbb{G}_1$ or $\mathbb{G}_2$, or both, and it comprises the following algorithms:

- $\mathsf{AGHO.Keygen}(1^\kappa, l) \to (sk, pk)$: On input security parameter κ and integer l indicating the size of the vectors of messages to be signed, randomly select $(u_1, \ldots, u_l, v) \leftarrow \mathbb{Z}_p^l$, and set $sk = (u_1, \ldots, u_l, v)$, $pk = (\widehat{U}_1, \ldots, \widehat{U}_l, V)$ with $\widehat{U}_i = \widehat{Q}^{u_i} \in \mathbb{G}_2$ and $V = P^v \in \mathbb{G}_1$. Output (sk, pk).
- $\mathsf{AGHO.Sign}(sk, m) \to \sigma$: On input secret key $sk = x$ and message $(M_1, \ldots, M_l) \in \mathbb{G}_1^l$, randomly pick $r \leftarrow \mathbb{Z}_p$ and output the following signature:

$$(\widehat{R} = \widehat{Q}^r, \widehat{S} = \widehat{R}^v, T = (P \prod_i M_i^{-u_i})^{1/r}).$$

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

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

– $\mathsf{AGHO.Verify}(pk, M_1, ..., M_l, \sigma) \rightarrow 0/1$: On input public key pk, vector of messages $(M_1, \ldots, M_l) \in \mathbb{G}_1^l$, and signature $\sigma = (\widehat{R}, \widehat{S}, T)$, output 1 if the following is satisfied:

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

Otherwise, output 0.

In our protocol, registration authority RA uses this SPS scheme to sign a pair of messages: the user's public key upk and her hashed tag $\Gamma = \mathbf{G}(\tau)$. More specifically, $\mathsf{DS.Keygen}(1^\kappa)$ calls $\mathsf{AGHO.Keygen}(1^\kappa, 2) \rightarrow (sk, pk)$ while $\mathsf{DS.Sign}(sk, upk \| \tau)$ first computes $\Gamma \leftarrow \mathbf{G}(\tau)$ and then calls $\mathsf{AGHO.Sign}(sk, upk, \Gamma) \rightarrow \sigma = (\widehat{R}, \widehat{S}, T)$. Correspondingly, $\mathsf{DS.Verify}(pk, upk, \tau, \sigma)$ computes $\Gamma \leftarrow \mathbf{G}(\tau)$ and returns the output of the execution $\mathsf{AGHO.Verify}(pk, upk, \Gamma, \sigma)$.

5.2 Our Tag-based Aggregate Signature (ATS)

Our tag-based aggregate signature is an enhanced aggregate BLS signature [5, 4] that signs a pair of messages and tags.

The scheme consists 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 $a \in \mathbb{Z}_p$, and output signature $\sigma = (\widehat{A}, B) \in \mathbb{G}_2 \times \mathbb{G}_1$ constructed as follows:

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

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

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

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

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

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

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

– $\mathsf{ATS.VerifyAgg}(\tau, \{pk_i\}_{i\in[1..n]}, \{m_i\}_{i\in[1..n]}, \sigma) \rightarrow 0/1$: 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.

Definition 1 (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 p.p.t. algorithm to output $e(P, \widehat{Q})^{xyz}$.*

Definition 2 (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 p.p.t. algorithm to output $e(P, \widehat{Q})^{xy}$.*

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

This proof of this theorem is found in Appendix B.

5.3 Signatures of knowledge

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 sign a a message m while proving possession of a set of attribute values $\{\mathfrak{a}_i\}_{i\in\mathbb{I}}$, a user $\mathcal{U}$ aggregates the corresponding tag-based signatures and produces a signature of knowledge for message m with respect to relation $\mathcal{R}$ defined as $(pp, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$ if and only if the following equalities hold:

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

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 thus in a more efficient signature of knowledge.

To reduce the computational cost incurred by the signature of knowledge, we use witness randomization 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 only 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:

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

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 $(pp, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$ if and only if the following hold:

$$\bar{P} = P^\delta \quad ; \quad \bar{V} = V^\delta \quad ; \quad H_i = \mathbf{H}(\mathfrak{a}_i)^\delta \quad ; \quad \bar{upk} = \bar{P}^{usk}. \tag{4}$$

Next, $\mathcal{U}$ produces a signature of knowledge η for $\mathcal{R}$, and correspondingly, sets her signature of knowledge of a valid credential ζ to $\zeta = (\eta, \bar{P}, \bar{V}, \widehat{R}', \widehat{S}', \bar{T}, \widehat{A}', \bar{B}, \bar{\Gamma}, \bar{upk}, \{\mathfrak{a}_i, H_i\}_{i \in \mathbb{I}})$.

A verifier $\mathcal{V}$ accepts ζ if $\bar{P} \neq 1$, η is a valid signature of knowledge for the relation depicted in equation 4 and the following equalities hold:

$$\begin{aligned} e(\bar{B}, \widehat{Q}) &= \prod_{i \in \mathbb{I}} e(H_i, \widehat{X}_i) \cdot e(\bar{\Gamma}, \widehat{A}') \; ; \\ e(\bar{P}, \widehat{Q}) &= e(\bar{T}, \widehat{R}') \cdot e(\bar{upk}, \widehat{U}_1) \cdot e(\bar{\Gamma}, \widehat{U}_2) \; ; \\ e(\bar{V}, \widehat{R}') &= e(\bar{P}, \widehat{S}'). \end{aligned} \tag{5}$$

This signature preserves the simulatability and extractability of signatures of knowledge. More details are given in Appendix C.

6 Evaluation

We evaluate the performance of our credential system instantiated using pairing-friendly cryptographic groups. All cryptographic operations are implemented over the BLS12-381 elliptic curve, though our construction is not tied to this choice and can be instantiated with other curves. We implement the credential system in Go using the `gnark-crypto` library [7]. Benchmarks were collected on a Macbook Pro with an M1 Pro chip and 16GB RAM.

To assess efficiency, we benchmark the cost of Sign and Verify while varying (i) the total number of attributes n, and (ii) the number of selectively revealed attributes k. Figures 2 and 3 display the results.

Our Sign procedure consists of two phases: setup captures the user preparing commitments to 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 scales with k. Verification also depends primarily on the number of revealed attributes.

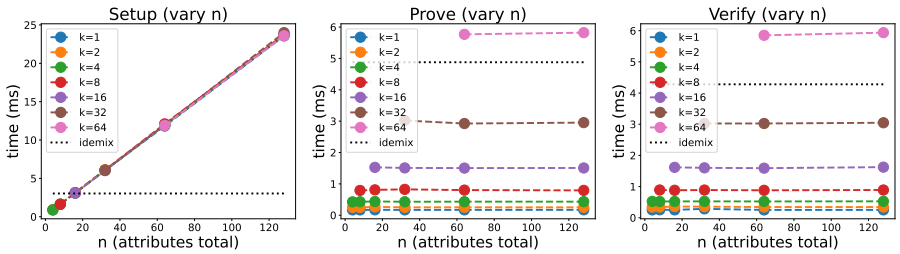

Fig. 2. Scaling of setup, proving, and verification with total attributes n varying, for fixed k.

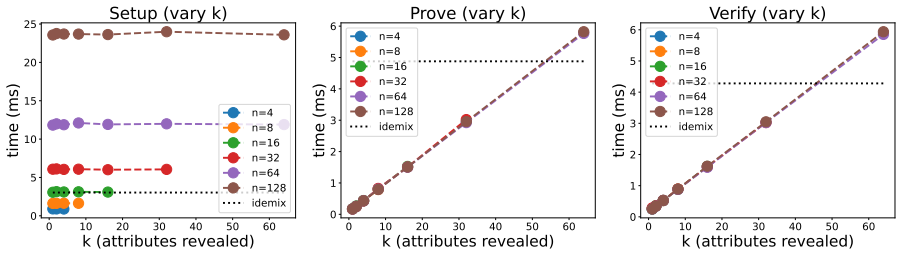

Fig. 3. Scaling of setup, proving, and verification with revealed attributes k varying, for fixed n.

As a baseline, we include a comparison with Idemix [1], plotted as a dotted horizontal reference line. Idemix always uses $n = 4$ attributes, and in every showing all attributes are hidden ($k = n = 4$). To aid readability, we plot its performance as a flat line across the range of n and k, rather than a single point. Idemix assumes a single-issuer setting, in contrast to our system which is designed in the multi-issuer setting. One protocol may be preferred to the other in practice due to the different assumptions on the functionality of the issuers.

Our evaluation shows that even for $n = 128$ attributes with $k = 64$ selectively revealed, both proving and verification complete under 10ms on a laptop. Setup scales linearly with n, while proving and verification scale linearly with k. Overall, the evaluation demonstrates the practical efficiency of our credential system. Both setup and showing costs are low enough for client devices, and verification remains lightweight.

7 Related Work

Single-Issuer Anonymous Credential Systems. Early work by Camenisch and Lysyanskaya introduced anonymous credential systems based on the strong-RSA assumption [10], enabling issuance of signatures on committed values. They later proposed a pairing-based construction in the random oracle model [11], though

signature size and verification complexity grow linearly with the number of attribute values signed. An extension [19] 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 [21] 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. They do not support decentralized issuance. More recently, [17] proposes structure-preserving signatures on equivalence classes. This facilitates very efficient, privacy-preserving proofs. The security of the scheme is proven in the generic group model. [22] presents an efficient scheme that allows for unlinkable proofs of knowledge over a subset of signed messages, building upon the Pointcheval-Sanders signature scheme. Its security is also proven in the generic group model.

Multi-Issuer Anonymous Credentials. To address centralization, multi-issuer anonymous credentials allow distinct authorities to issue credentials independently. These schemes often suffer from linear complexity in the number of issuers, and frequently reveal the set of participating issuers during credential showing. This weakens privacy: credential showings are only unlinkable among users with the same attributes and the same issuer set.

To work towards addressing this, recent work introduced *issuer-hiding* credentials [3, 6, 16, 23]. These schemes operate in the classical single-authority setting. Although multiple different issuers are able to issue each credential, forming the anonymity set for the issuers of each credential showing, the expectation is that credentials from multiple issuers will be shown separately.

The system in [20] is the first issuer-hiding scheme in the multi-issuer setting. Achieving issuer-hiding introduces additional complexity in comparison to our scheme. Of their two schemes, one of them does not have fixed-size credentials, and the other achieves constant-sized credentials but requires a trusted setup to do so. Multi-issuer credentials address the trust centralization problem, but introduce new challenges in efficiency and privacy. These gaps have motivated the development of aggregate credential systems, aiming to combine multiple independent credentials into a compact credential showing.

Aggregate Anonymous Credentials. Aggregate credentials aim to combine independently issued credentials into a single, compact showing, while maintaining unlinkability. Canard and Lescuyer [12] 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. Later work [8] achieves efficient unlinkable credential showings over subsets of messages for multiple issuers, but relies on relatively strong cryptographic assumptions and requires a trusted setup. Hebant and Pointcheval [18] 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. They also show that an earlier Canard-Lescuyer construction [13] fails to achieve unlinkability.

8 Conclusion

We introduced what we call *aggregate tag-based signatures* that enable a user to aggregate signatures produced by different issuers 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 benchmark results demonstrate practical performances, highlighting the scheme's potential for real-world decentralized identity applications.

References

1. `https://hyperledger-fabric.readthedocs.io/en/release-2.2/idemix.html`.
2. 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.
3. Jan Bobolz, Fabian Eidens, Stephan Krenn, Sebastian Ramacher, and Kai Samelin. Issuer-hiding attribute-based credentials. In *Cryptology and Network Security: 20th International Conference, CANS 2021, Vienna, Austria, December 13-15, 2021, Proceedings 20*, pages 158–178. Springer, 2021.
4. 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.
5. 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.
6. Daniel Bosk, Davide Frey, Mathieu Gestin, and Guillaume Piolle. Hidden issuer anonymous credential. *Proceedings on Privacy Enhancing Technologies*, 2022:571–607, 2022.
7. Gautam Botrel, Thomas Piellard, Youssef El Housni, Arya Tabaie, Gus Gutoski, and Ivo Kubjas. Consensys/gnark-crypto: v0.15.0, January 2025.
8. 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.

9. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT*, pages 93–118, 2001.
10. 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.
11. 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.
12. 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.
13. Sébastien Canard and Roch Lescuyer. Protecting privacy by sanitizing personal data: a new approach to anonymous credentials. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pages 381–392, 2013.
14. 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.
15. David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.
16. Aisling Connolly, Pascal Lafourcade, and Octavio Perez Kempner. Improved constructions of anonymous credentials from structure-preserving signatures on equivalence classes. In *IACR International Conference on Public-Key Cryptography*, pages 409–438. Springer, 2022.
17. 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.
18. 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.
19. 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.
20. Omid Mir, Balthazar Bauer, Scott Griffy, Anna Lysyanskaya, and Daniel Slamanig. Aggregate signatures with versatile randomization and issuer-hiding multi-authority anonymous credentials. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 30–44, 2023.
21. David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, *Proceedings of the Cryptographers Track at the RSA Conference*, volume 9610 of *LNCS*, pages 111–126. Springer, 2016.
22. 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.

23. Olivier Sanders and Jacques Traoré. Compact issuer-hiding authentication, application to anonymous credential. *Proceedings on Privacy Enhancing Technologies*, 2024.

A Proof of Theorem 1

Theorem. *If* DS *is a digital signature scheme that is an existentially unforgeable,* ATS *is a signature scheme that is existentially unforgeable under tag-based aggregation, and* SoK *is signature of knowledge scheme that satisfies the properties of extractability and simulatability, then protocol Π described in Section 4 securely realizes ideal functionality $\mathcal{F}$.*

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 $\mathcal{F}$. To that end, we leverage the hybrid argument and define a sequence of *indistinguishable games* that start with *Game 0* representing the real world, and end with *Game 10* 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 her 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 $\mathsf{Register}$ 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 $(\mathsf{SignEND}, \vec{\mathfrak{a}}, m, \zeta)$ from $\mathcal{F}'$ for $\vec{\mathfrak{a}} = (\mathfrak{a}_1, \ldots, \mathfrak{a}_N)$, $\mathcal{S}$ 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 $\zeta^\star \leftarrow \mathsf{SoK.SimSign}(td, \mathbb{x}, m, pp)$, which yields $\mathsf{SoK.Verify}(\mathbb{x}, \zeta^\star, pp)$ to return $\mathtt{true}$.

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 $\mathsf{SoK.Verify}$, 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 $\mathsf{Register}$ 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 $(\vec{\mathfrak{a}} = (\mathfrak{a}_1, ..., \mathfrak{a}_N), m, \zeta)$ that a user $\mathcal{U}$ presents, in the real world, to an honest verifier V, $\mathcal{S}$ (simulating V) calls $\mathsf{bool} \leftarrow \mathsf{SoK.Verify}(pp, \vec{\mathfrak{a}}, m, \zeta)$. We recall that $\mathfrak{a}_i = \varnothing$ if the attribute value is not disclosed.

If $\mathsf{bool} = \mathtt{false}$, $\mathcal{S}$ sends $(m, \vec{\mathfrak{a}} = (\mathfrak{a}_1, ..., \mathfrak{a}_N), \zeta)$ to $\mathcal{F}'$, which returns $(\mathsf{Verified}, \mathsf{bool}^\star)$. Two scenarios present themselves. **(1)** There exists a dishonest user with attribute values $\vec{\mathfrak{a}'} = (\mathfrak{a}'_1, ..., \mathfrak{a}'_N)$ 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 ζ 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 $(\mathsf{Sign}, \vec{\mathfrak{a}}, \sigma, m)$.
  - If $\mathcal{F}'$ aborts, then $\mathcal{S}$ sends request $(\mathsf{Verify}, V, \vec{\mathfrak{a}}, \zeta)$ to $\mathcal{F}'$.
  - Otherwise, $\mathcal{F}'$ returns $(\mathsf{SignEND}, \zeta^\star)$, and $\mathcal{S}$ sends request $(\mathsf{Verify}, V, \vec{\mathfrak{a}}, \zeta^\star)$ 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 Sign call, $\mathcal{F}'$ will not abort and will return $(\mathsf{SignEND}, \zeta^\star)$, and the subsequent call $(\mathsf{Verify}, V, \vec{\mathfrak{a}}, \zeta^\star)$ 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 Sign, and when $\mathcal{S}$ calls $\mathcal{F}'$ with $(\mathsf{Verify}, V, \vec{\mathfrak{a}}, \zeta)$, 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 ζ, 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 ζ, 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 signatures 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

Theorem. *Our tag-based aggregate signature is existentially unforgeable, in the random oracle model, under tag-based aggregation under the BCDH and m-BCDH assumptions.*

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 $\mathcal{A}$ who is able to succeed in said experiment, i.e., forge an

aggregate signature $(\widehat{A}, B)$ for some message $(m_i^*)_{i \in \mathbb{I} \subset [1..N]}$ and tag τ^* that successfully verifies the following equation:

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

and $\exists j \in \mathbb{I}$ s.t. $\mathcal{A}$ does not hold the secret key underlying public key pk_j and she has never requested a tag-based signature for 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, she 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, she 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, she 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 Signature of Knowledge

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 $i \in \mathbb{I}$. $\mathcal{U}$ signs a message m while proving possession of valid credential and attributes $\{\mathfrak{a}_i\}_{i\in\mathbb{I}}$ 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 $(pp, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$ iff

$$\bar{P} = P^\delta \; ; \; \bar{V} = V^\delta \; ; \; H_i = \mathbf{H}(\mathfrak{a}_i)^\delta \; ; \; \bar{upk} = \bar{P}^{usk} \tag{6}$$

$\mathcal{U}$ now computes a signature of knowledge η for $\mathcal{R}$,and lets her signature of knowledge of a valid credential be $\zeta = (\eta, \bar{P}, \bar{V}, \widehat{R}', \widehat{S}', \bar{T}, \widehat{A}', \bar{B}, \bar{\Gamma}, \bar{upk}, \{\mathfrak{a}_i, H_i\}_{i\in\mathbb{I}})$.

A verifier accepts ζ if $\bar{P} \neq 1$, η is a valid signature of knowledge for the relation depicted in equation 4 and the following equalities hold:

$$e(\bar{P}, \widehat{Q}) = e(\bar{T}, \widehat{R}') \cdot e(\bar{upk}, \widehat{U}_1) \cdot e(\bar{\Gamma}, \widehat{U}_2) \; ; \; e(\bar{V}, \widehat{R}') = e(\bar{P}, \widehat{S}') \; ; \tag{7}$$
$$e(\bar{B}, \widehat{Q}) = \prod_{i\in\mathbb{I}} e(H_i, \widehat{X}_i) \cdot e(\bar{\Gamma}, \widehat{A}').$$

Next, we describe how η is computed and show that our signature of knowledge satisfies the properties of extractability and simulatability.

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 that she knows a valid witness $\mathbb{w} = (\delta, usk)$ for the relation depicted in equation 6, 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 m, 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 $c = \mathcal{H}_p(A, B, \{C_i\}_{i\in\mathbb{I}}, D, m)$, and outputting $\eta = (A, B, \{C_i\}_{i\in\mathbb{I}}, D, \pi_1, \pi_2)$.

C.2 Extractability

We recall that our signature of knowledge on some message m is defined as tuple $\zeta = (\eta, \bar{P}, \bar{V}, \widehat{R}', \widehat{S}', \bar{T}, \widehat{A}', \bar{B}, \bar{\Gamma}, \bar{upk}, \{\mathfrak{a}_i, H_i\}_{i\in\mathbb{I}})$.

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 5, 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 signature of knowledge 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

In this section, we assume that the decisional Diffie-Hellman (DDH) assumption holds in $\mathbb{G}_1$.

Definition (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.*

We now show how to simulate a valid signature ζ 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 $\widehat{A}' = (\widehat{Q} \prod_{i\in\mathbb{I}} \widehat{ipk}_i^{-\eta_i})^{\delta/\nu}$ (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 7);
- 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 $\mathcal{H}_p(A, B, \{C_i\}_{i\in\mathbb{I}}, D, m)$ as value c;
- Finally, set η to tuple $(A, B, \{C_i\}_{i\in\mathbb{I}}, D, \pi_1, \pi_2)$.