Protego: A Credential Scheme for Permissioned Blockchains (Extended Version)

Aisling Connolly[1], Jerome Deschamps[2],
Pascal Lafourcade[2], and Octavio Perez Kempner[3,4]

[1] DFINITY*
aislingmconnolly@gmail.com
[2] LIMOS, University Clermont Auvergne, France
jerome.deschamps@etu.uca.fr
pascal.lafourcade@uca.fr
[3] DIENS, École normale supérieure, CNRS, PSL University, Paris, France
[4] be-ys Research, France
octavio.perez.kempner@ens.fr

Abstract. Recent works to improve privacy and auditability in permissioned blockchains like Hyperledger Fabric rely on Idemix, the only anonymous credential system that has been integrated to date. The current Idemix implementation in Hyperledger Fabric (v2.4) only supports a fixed set of attributes, it does not support revocation features, nor does it support anonymous endorsement of transactions (in Fabric, transactions need to be approved by a subset of peers before consensus). A prototype Idemix extension by Bogatov *et al.* (CANS, 2021) was proposed to include revocation, auditability, and to gain privacy for users. We explore how to gain efficiency, functionality, and further privacy departing from recent works on anonymous credentials based on Structure-Preserving Signatures on Equivalence Classes. As a result, we propose Protego and Protego Duo, two alternatives for Idemix and its recent extensions. We discuss how they can be used in the permissioned blockchain setting and integrated to Hyperledger Fabric. We also provide a prototype implementation and benchmarks showing that both alternatives are twice as fast as state-of-the-art-approaches.

Keywords: anonymous credentials, auditability, Hyperledger Fabric, mercurial signatures, permissioned blockchains, SPS-EQ.

1 Introduction

When first introduced, the core use of blockchains was in the *permissionless* setting; anyone could join and participate. Over the years, blockchains have also found use within consortiums, where several authorized organizations wish to share information among the group, but not necessarily to the public as a whole. This need gave rise to *permissioned* blockchains whereby authorities

* Work done while the author was at Wordline Global.

are established to define a set of participants. When a federation of authorities (*consortium*), each in control of a subset of participants, shares the blockchain's governance, the term *federated* is also used to describe such blockchains.

The use of federated blockchains increased to address the need to run a common business logic within a closed environment. As an example, one can consider pharmaceutical companies that would like to trade sensitive information about product developments and agree on supplies or prices in a consortium with partial trust. A recurrent problem in such scenarios is that of privacy while being compliant with regulations and *Know Your Customer* practices. Agreeing with other entities to run a shared business logic should not imply that everything needs be public within the consortium. Privacy still needs to be provided without affecting existing regulations, *e.g.,* when considering bilateral agreements.

The most developed permissioned platform is Hyperledger Fabric (or simply Fabric). By default, it provides no privacy features as everything (users and transactions) within a given federation is public. Motivated by the need to protect business interests and to meet regulatory requirements, some privacy features were integrated, notably, the Identity Mixer [19, 10] (or Idemix for short). This anonymous attribute-based credential (ABC) scheme gave the first glimpse of privacy for users within a consortium. The current Idemix integration with Fabric is still quite limited in efficiency, functionality, and privacy levels. Recently, a prototype extending Idemix has emerged [7], incorporating revocation features, the possibility of auditing transactions, and higher levels of privacy for the user. This extension is based on *delegatable credentials* but it still suffers some inherent limitations, which puts all trust on a root certificate authority and requires the generation of many zero-knowledge proofs to sign a transaction.

Recent results introduced newer models to build ABC's, providing a host of extra functionalities and more efficient constructions. The main goal of this work is to leverage such results, to position them in the blockchain scenario and provide an alternative to Idemix (and its extension) in a bid to overcome existing privacy and functional limitations, while also improving efficiency.

Contributions. We explore alternative mechanisms to build a practical ABC. First, we extend recent works [14, 11] based on Structure-Preserving Signatures on Equivalence Classes (SPS-EQ) to support audibility features, while also integrating the revocation ideas from [13]. Such extension relies on the random oracle model (already present in the blockchain setting) to generate non-interactive showing proofs. We also present and discuss two alternatives to the use of delegatable credentials to hide the identity of credential issuers, following the formalizations from [11] and [6] but using new approaches.

As a result, we build Protego, a new ABC for permissioned blockchains. We also present Protego Duo, a variant based on a different approach to hide the identity of credential issuers. Both support revocation and auditing features, which are important to enable a broader variety of use cases for permissioned blockchains. We discuss how to integrate our work with Fabric, compare it with Idemix and its recent extensions, and provide a prototype implementation showing that Protego and Protego Duo are faster than the latest Idemix extension.

Related work. We describe the related work following two main streams; the results addressing privacy concerns in Fabric, and parallel research developments.

Privacy concerns in Fabric. The most closely related work appears with the introduction of Idemix [19] and its extension to include revocation and auditability [7]. Adding auditability is crucial for permissioned blockchains as they are often used in heavily regulated industries. Privacy-preserving auditing for distributed ledgers was introduced in [17] under the guise of zkLedger. This general solution offered great functionality in that it provided confidentiality of transactions, and privacy of the users within the transaction. However, it assumed low transaction volume between few participants and as such is quite limited in scalability. Fabric-friendly auditable token payments were introduced in [2] and were based on threshold blind signatures. The core idea to achieve auditability was to encrypt the user's public key under the public key of an auditor. This is the same approach in [7], which we also use in this work. Although the auditing ideas are similar, the construction pertains solely to transaction privacy and offers no identity privacy for a user. Following the approach of gaining auditability of transactions, auditable smart contracts were captured by FabZK [15] which is based on Pedersen commitments and zero-knowledge proofs. To achieve auditability, the structure of the ledger is modified, and as such, would need to make considerable changes to existing used permissioned blockchain platforms.

In Fabric, the validity of a transaction is established by obtaining *endorsements* from peers in the network. One of the limitations in Idemix and its extension is the lack of privacy or anonymity for endorsing peers. A potential solution to this was proposed in [16], where the endorsement policy is based on a ring signature scheme such that the endorsement set itself is not revealed, but only that sufficiently many endorsement signatures were obtained. Another approach to obtain privacy-preserving endorsements was described in [3], leveraging Idemix credentials to gain endorser-privacy, and as such, inherits the limitations (notably leaking the endorser's organization) that come with Idemix.

Attribute-based credentials. Early anonymous credential schemes were built from blind signatures, whereby a user obtained a blind signature from an issuer on the user's commitment to its attributes. When the user later authenticates, they provide the signature, the shown attributes, and a proof of knowledge of all unshown attributes. These schemes are limited as they can only be shown once. Subsequent work like the one underlying Idemix [9] allowed for an arbitrary number of unlinkable showings. A user obtains a signature on a commitment on attributes, randomizes the signature, and proves in zero-knowledge that the randomized signature corresponds to the shown and unshown attributes.

Recent work from [14] circumvented inefficiencies in the above ideas by coining two new primitives: set-commitment schemes, and SPS-EQ. As a result, authors obtained a scheme allowing to randomize both the signature and the commitment on a set of attributes. Furthermore, a subset-opening of the set-commitment yielded constant-size selective showing of attributes.

New work from [11] extended [14], improving the expressivity, efficiency trade-offs and introducing the notion of *signer-hiding* (also known as *issuer-*

hiding [6]) to allow users to easily randomize the public key used to generate a signature to hide the identity of credential issuers. Authors achieve the previous points using a *Set-Commitment scheme supporting Disjoint Sets* (SCDS) and mercurial signatures. The latter primitive extends SPS-EQ to consider equivalence classes not only on the message space but also on the key space.

We build on top of the above-mentioned previous works but unlike [11], we work with the generic group model as our main motivation is the proposal of efficient alternatives. Therefore, we use the mercurial signature scheme from [12].

2 Privacy Notions in Fabric

By default, Fabric does not provide any privacy-preserving feature; reading the blockchain anyone can know (1) who triggered a smart contract using which arguments (transaction proposals are signed by the clients), (2) who vouched for its execution (endorsers also sign their responses) concerning reading and writing sets; and (3) why a given transaction was marked as invalid (either because of invalid read/write sets or because the endorsement policy check failed). Furthermore, checking access control and endorsement policies links different organizations, users and their attributes to concrete actions on the system.

Such limitations severely restrict the use of Fabric. From the user perspective this impacts the enforcement of different regulations. For organizations, the case is similar. Consider a consortium of pharmaceuticals organizations that run a common business logic to exchange information on medical research. If the entity behind a request is known, other organizations can infer (based on the request) which drug the entity in question is trying to develop. If the endorsers are known, information about who executes what can disclose business relations.

Idemix, its limitations, and extensions. Several proposals aimed to address the aforementioned limitations but only Idemix has been integrated to Fabric so far. Idemix allows a Membership Service Provider (MSP) to issue credentials enabling users to sign transactions anonymously. In brief, users generate a zero-knowledge proof attesting that the MSP issued them a credential on its attributes to sign a transaction. Fabric's support for Idemix was added in v1.3, providing the first solution to tackle the problem of *participant* privacy. Unfortunately, as for v2.4 the Idemix implementation still suffers severe limitations:

1. It supports a fixed set of only four attributes.
2. It does not support revocation features.
3. Credentials leak the MSP ID, meaning that anonymity is local to users within an organization. For this reason, current deployments can only use a single MSP for the whole network, introducing a single point of failure.
4. It does not support the issuance of Idemix credentials for the endorsing peers, meaning that the identity of endorsers is always leaked.

The most promising effort to extend the functionality of Idemix appeared in [7]. Their aim was to extend the original credential system to support delegatable credentials, while integrating revocation and auditability features (solving three of the four limitations). Below we outline the main ideas introduced in [7].

Delegatable Credentials. In a bid to overcome the issue of Idemix credentials leaking the MSP ID and thus the affiliation of the user, a trusted root authority provides credentials to intermediate authorities. This way users can obtain credentials from intermediate authorities. To sign a transaction, the user must generate a zero-knowledge proof attesting that (1) the signer owns the credential; (2) the signature is valid; (3) all adjacent delegation levels are legitimate; and (4) that the top-level public key belongs to the root authority.

Revocation and Auditability. To generate efficient proofs of non-revocation, the system timeline is divided into *epochs*. Issued credentials are only valid for a given epoch, and must be reissued as the timeline advances. For each epoch, a user requests a revocation handler that binds their public key to the epoch. When presenting a credential, the user also provides a proof of non-revocation. To enable auditing of a transaction, users verifiably encrypt their public key under an authorized auditor's public key.

To date, some functionalities remain limited. (1) There is still no notion of privacy for endorsers. (2) Delegatable credentials require proving knowledge of a list of keys. (3) The root authority is still a single point of failure. (4) Selective disclosure of attributes requires computation linear in the size of all the attributes encoded in the credential. (5) Many zero-knowledge proofs need to be generated for each transaction. (6) Many pairings need to be computed for verification.

3 Protego

To build an ABC scheme that overcomes the inherent limitations from Idemix and its extension, we argue that changing some of the underlying building blocks is necessary. We take the framework from [14] as our starting point, incorporate the recent improvements from [11], and include the revocation extension originally proposed in [13]. Below, we walk through the different building blocks and build an argument for how and why these components yield greater functionality and efficiency for a credential system in the permissioned blockchain setting.

Notation. Let BGGen be a p.p.t algorithm that on input 1^λ with λ the security parameter, returns a description $\mathsf{BG}{=}(p,\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,P_1,P_2,e)$ of an asymmetric (Type-3) bilinear group where $\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T$ are cyclic groups of prime order p with $\lceil \log_2 p \rceil = \lambda$, P_1 and P_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable (non-degenerate) bilinear map. For all $a \in \mathbb{Z}_p$, $[a]_s = aP_s \in \mathbb{G}_s$ denotes the implicit representation of a in $\mathbb{G}_s$ for $s \in \{1, 2\}$. For vectors $\mathbf{a}, \mathbf{b}$ we extend the pairing notation to $e([\mathbf{a}]_1, [\mathbf{b}]_2) := [\mathbf{ab}]_T \in \mathbb{G}_T$. $r \xleftarrow{\$} \mathcal{S}$ denotes sampling r from set $\mathcal{S}$ uniformly at random. $\mathcal{A}(x; y)$ indicates that y is passed directly to $\mathcal{A}$ on input x. Hash functions are denoted by $\mathcal{H}$.

SCDS. Using commitment schemes that allow to commit to *sets* of attributes enables constant-size openings of subsets (selective disclosure) of the committed sets. These schemes support commitment randomization without the need to rely on zero-knowledge proofs of correct randomization, as the corresponding witness for openings can be adapted accordingly with respect to the randomization of the committed set. The set-commitment scheme presented in [11] extends [14] to support openings of attribute sets *disjoint* from the committed set. This is particularly useful in the permissioned blockchain setting, *e.g.,* to model access control policies. Furthermore, the scheme from [11] also supports the use of proof of exponentiations (PoE) to outsource some of the computational cost from the verifier to the prover. In the case of Fabric, this is a particularly interesting feature to make the endorser's verification faster when validating a transaction proposal. We recall the syntax and security properties for SCDS in Appendix A.

Mercurial Signatures. The introduction of SPS-EQ in [14] allowed to adapt a signature on a representative message to a signature on a different representative (in a given equivalence class) without knowledge of the secret key. If the adapted signature is indistinguishable from a fresh signature on a random message, the scheme satisfies the notion of *perfect adaption*. This, together with the randomizability of the set-commitment scheme, allows to consistently and efficiently update the signature of a credential, bypassing the need to generate and keep account of pseudonyms and NIZK proofs that are required in all previous works based on Idemix. Our approach here is to use mercurial signatures (see Appendix B for syntax and security properties) to also randomize the correspondng public keys while consistently adapting the signatures.

ABC model. We can rely on the random oracle model and apply the Fiat-Shamir transform to the ABC scheme from [11] (the showing protocol is a three move public coin one). However, in the previous ABC interaction is required in the showing protocol to provide freshness (*i.e.,* to avoid replay attacks). To overcome this issue we require the user to send the transaction proposal during the first move. Thus, applying the Fiat-Shamir transform to the first move bounds the credential showing to that particular transaction so that it cannot be replayed. The modifications when compared to [11] are as follows: (1) we adapt the model to non-interactive showings, (2) we keep the SCDS scheme from [11] as it is but replace the signature scheme with the one given in [12], (3) we define a revocation authority as in [13], and an auditing authority in the model (not considered in the previous works), and (4) we build a *malleable* NIZK argument that can be pre-computed to obtain a more efficient issuer-hiding feature.

We now present our ABC scheme *Protego* introducing the syntax first. Then we elaborate on the revocation, auditing and issuer-hiding approaches. Finally, we discuss our construction and the integration with Fabric.

ABC Syntax. An ABC consists of the following p.p.t algorithms:

$\mathsf{Setup}(1^\lambda, \mathsf{aux})$ takes a security parameter λ and some optional auxiliary information aux (which may fix an universe of attributes, attribute values and other parameters) and outputs public parameters pp discarding any trapdoor.

$\mathsf{TSetup}(1^\lambda, \mathsf{aux})$ like Setup but returns a trapdoor.

$\mathsf{OKGen}(\mathsf{pp})$ takes pp and outputs an organization key pair $(\mathsf{osk}, \mathsf{opk})$.

$\mathsf{UKGen}(\mathsf{pp})$ takes pp and outputs a user key pair $(\mathsf{usk}, \mathsf{upk})$.

$\mathsf{AAKGen}(\mathsf{pp})$ takes pp and outputs an auditor key pair $(\mathsf{ask}, \mathsf{apk})$.

$\mathsf{RAKGen}(\mathsf{pp})$ takes pp and outputs a revocation key pair $(\mathsf{rsk}, \mathsf{rpk})$.

$\mathsf{Obtain}(\mathsf{pp}, \mathsf{usk}, \mathsf{opk}, \mathsf{apk}, \mathcal{X}, \mathsf{nym})$ and $\mathsf{Issue}(\mathsf{pp},\mathsf{upk},\mathsf{osk},\mathsf{apk}, \mathcal{X},\mathsf{nym})$ are run by a user and the organization respectively, who interact during execution. Obtain takes pp, the user's secret key usk, an organization's public key opk, an auditor's public key apk, an attribute set $\mathcal{X}$ of size $|\mathcal{X}| < t$, and a pseudonym nym used for revocation. Issue takes pp, a public key upk, a secret key osk, an auditor's public key apk, an attribute set $\mathcal{X}$ of size $|\mathcal{X}| < t$, and a pseudonym nym. At the end of this protocol, Obtain outputs a credential cred on $\mathcal{X}$ for the user or $\perp$ if the execution failed.

$\mathsf{Show}(\mathsf{pp}, \mathsf{opk}, \mathsf{upk}, \mathsf{usk}, \mathsf{cred}, \mathcal{X}, \mathcal{S}, \mathcal{D}, \mathsf{aux})$ takes pp, a public key opk, a key pair $(\mathsf{usk}, \mathsf{upk})$, a credential cred for the attribute set $\mathcal{X}$, potentially non-empty sets $\mathcal{S} \subseteq \mathcal{X}$, $\mathcal{D} \not\subseteq \mathcal{X}$ representing attributes sets being a subset $(\mathcal{S})$ or disjoint $(\mathcal{D})$ to the attribute set $(\mathcal{X})$ committed in the credential, and auxiliary information aux. It outputs a proof π.

$\mathsf{Verify}(\mathsf{pp}, \mathsf{opk}, \mathcal{X}, \mathcal{S}, \mathcal{D}, \pi, \mathsf{aux})$ takes pp, the (potentially empty) sets $\mathcal{S}$ and $\mathcal{D}$, a proof π and auxiliary information aux. It outputs 1 or 0 indicating whether the credential showing proof π was accepted or not.

$\mathsf{RSetup}(\mathsf{pp}, (\mathsf{rsk}, \mathsf{rpk}), \mathsf{NYM}, \mathsf{RNYM})$ takes pp, a revocation key pair $(\mathsf{rsk}, \mathsf{rpk})$ and two disjoint lists NYM and RNYM (holding valid and revoked pseudonyms). It outputs auxiliary information $\mathsf{aux}_{\mathsf{rev}}$ for the revocation authority and revocation information $\mathbb{R} = (\mathbb{R}_V, \mathbb{R}_S)$. $\mathbb{R}_V$ is needed for verifying the revocation status and $\mathbb{R}_S$ is a list holding the revocation information per nym.

$\mathsf{Revoke}(\mathsf{pp}, (\mathsf{rsk}, \mathsf{rpk}), \mathsf{aux}_{\mathsf{rev}}, \mathbb{R}, b)$ takes pp, $(\mathsf{rsk}, \mathsf{rpk})$, $\mathsf{aux}_{\mathsf{rev}}$, $\mathbb{R}$ and a bit b indicating revocation/unrevocation. It outputs information $\mathbb{R}'$ and $\mathsf{aux}'_{\mathsf{rev}}$.

$\mathsf{AuditEnc}(\mathsf{upk}, \mathsf{apk})$ takes upk and apk. It outputs an encryption Enc of upk under apk and auxiliary information α.

$\mathsf{AuditDec}(\mathsf{Enc}, \mathsf{ask})$ takes Enc and ask. It outputs a decryption of Enc using ask.

$\mathsf{AuditPrv}(\mathsf{Enc}, \alpha, \mathsf{usk}, \mathsf{apk})$ takes Enc, α, usk, and apk. It generates a proof for Enc being the encryption of upk under apk and outputs a proof π.

$\mathsf{AuditVerify}(\mathsf{apk}, \pi)$ takes apk and a proof π for the correct encryption of a user's public key under apk and outputs 1 if and only if the proof verifies.

Revocation. We opt to integrate the work from [13] as pointed out in [11]. The revocation system from [13] defines a revocation authority responsible for managing a white and a blacklist of revocation handlers. The authority publishes an accumulator RevAcc representing the blacklist, and maintains a public list of non-membership witnesses for unrevoked users. During the issuing protocol, users are given a revocation handler that is encoded in the credential. To prove that they are not revoked during a showing, the user consistently randomizes its credential with the accumulator and the corresponding non-membership witness. Then the verifier checks that the (randomized) witness is valid for the revocation handler (encoded in the user credential), and with respect to the (randomized) accumulator. To work, the user must compute a Zero-Knowledge Proof of Knowledge (ZKPoK) on the correct randomization of the non-membership witness and the accumulator. As explained in [13], the revocation handler encoded in the user's credential is of the form $\mathsf{usk}_2(b+\mathsf{nym})P_1$, where usk_2 is an additional user secret key required for anonymity and nym is the pseudonym used for revocation. For this reason, users are required to manage augmented keys of the form $\mathsf{upk} = (\mathsf{upk}_1, \mathsf{upk}_2)$, $\mathsf{usk} = (\mathsf{usk}_1, \mathsf{usk}_2)$. Furthermore, for technical reasons, another component $\mathsf{usk}_2 Q$, where Q is a random element in $\mathbb{G}_1$ with unknown discrete logarithm, must be included in the credential.

Auditability. A credential in [14, 11] and [13] contains a tuple (C, rC, P_1) where C is the set commitment on the user attributes, r is a random value used for technical purposes and P_1 is used to compute a ZKPoK of the randomizer μ in $(\mu C, \mu rC, \mu P_1)$ during a showing. We borrow the idea of using a verifiable variant of ElGamal from [7] to prove the well-formedness of a ciphertext (encrypting the user's public key) with respect to the auditor's key. Therefore, we add the user's public key upk_1 and the auditor's public key apk as the sixth and seventh components to the credential. Thus, we now have revocable credentials of the form $(C,\ rC, P_1,\ \mathsf{usk}_2(b+\mathsf{nym})P_1,\ \mathsf{usk}_2 Q,\ \mathsf{upk}_1, \mathsf{apk})$, which can be randomized to obtain a tuple $(\mu C,\ \mu rC,\ \mu P_1,\ \mu\mathsf{usk}_2(b+\mathsf{nym})P_1,\ \mu\mathsf{usk}_2 Q,\ \mu\mathsf{usk}_1 P_1,\ \mu\mathsf{apk})$. We exploit this fact to allow the user to generate an *audit* proof that can be publicly verified without leaking information about the user's public key. This way, verifiers can check a proof using the sixth and seventh component in the credential to be sure that (1) the user encrypted a public key for which it has the corresponding secret key, and (2) using the correct one. Since the issuing authority signs the credential, the randomization needs to be consistent. Modifications required to implement our auditability approach are as follows:

1. The user randomizes its credential as usual to obtain a new one of the form $(C'_1,\ C'_2,\ C'_3,\ C'_4,\ C'_5,\ C'_6,\ C'_7){=}(\mu C_1,\ \mu C_2,\ \mu P_1,\ \mu C_4,\ \mu C_5,\ \mu\mathsf{upk}_1,\ \mu\mathsf{apk})$. Since only the user knows the randomizer μ, its public key remains hidden.
2. The user picks $\alpha \in \mathbb{Z}_p$ and encrypts its own public key using ElGamal encryption with auditor's public key apk and randomness α to obtain a ciphertext $\mathsf{enc} = (\mathsf{enc}_1, \mathsf{enc}_2) = (\mathsf{upk}_1 + \alpha\mathsf{apk}, \alpha P_1)$.
3. The user runs the algorithm $\mathsf{AuditPrv}$ (Figure 1) with input $(\mathsf{enc}, \alpha, \mathsf{usk}_1, \mathsf{apk})$ to obtain c, z_1 and z_2.
4. Then, the user picks $\beta \leftarrow \mathbb{Z}_p$, computes $t_1 = \beta P_2$, $t_2 = \beta\mu P_2$, $t_3 = \alpha\beta P_2$ and sends $(\mathsf{enc}, c, z_1, z_2, t_1, t_2, t_3)$ to the verifier alongside the randomized credential from step 1.
5. The verifier checks the well-formedness of the ElGamal encryption pair running the algorithm $\mathsf{AuditVerify}$ (Figure 1) with input $(c, \mathsf{enc}, z_1, z_2)$. If the check succeeds, it checks the following pairing equations to verify that the encrypted public key is the one in the credential:
$e(\mathsf{enc}_2, t_2){=}e(C'_3, t_3) \wedge e(\mathsf{enc}_2, t_1){=}e(P_1, t_3) \wedge e(\mathsf{enc}_1, t_2){=}e(C'_6, t_1){+}e(C'_7, t_3)$

Observe that the verifier knows $\mu P_1 = C'_3$, $\mu\mathsf{usk}_1 P_1 = C'_6$, $\mu\mathsf{ask} P_1 = C'_7$, $(\mathsf{usk}_1 + \alpha\mathsf{ask})P_1 = \mathsf{enc}_1$, $\alpha P_1 = \mathsf{enc}_2$, $\beta P_2 = t_1$, $\beta\mu P_2 = t_2$ and $\alpha\beta P_2 = t_3$. With β the user is able to randomize the other values so that the pairing equation can be checked to verify the relation between the ElGamal ciphertext and the randomized public key in C'_6, without leaking information about the user's public key. Furthermore, the first two pairing equations verify the well-formedness of t_1, t_2 and t_3 with respect to the user's credential and the ciphertext. Hence, the verifier will not be able to recover the user's public key nor the user cheat.

The proposed solution only adds two elements to the credential, while requiring the user to send two more elements in $\mathbb{G}_1$, three in $\mathbb{Z}_p$ and three in $\mathbb{G}_2$, for a total of eight. Computational cost remains low as it just involves the computation of seven pairings, the ElGamal encryption and two Schnorr proofs [18].

Issuer-hiding. In [11], since users can consistently randomize the signature on their credential and the issuer's public key, a fully adaptive NIZK argument is used to prove that a randomized issuer key belongs to the equivalence class of one of the keys contained in a list of issuers keys. This way, the randomized issuer key can be used to verify the credential while hiding the issuer's identity (like in a ring signature). In permissioned blockchains where there are multiple organizations that issue credentials, such a NIZK allows users holding valid signatures to pick any subset of issuer's public keys to generate a proof. In the extended version of this work (Appendix D), we adapt the previously mentioned proof system to the signature used, and make it malleable so that users can compute the proof once and then adapt it during showings with little computational cost.

Another approach following the work from [6] (briefly discussed in [11]) is to consider issuer-policies. An issuer-policy is a set $\{(\sigma_i, \mathsf{opk}_i)_{i\in[n]}\}$ of signatures on issuer's public keys generated by some verification secret key vsk. To hide the identity of an issuer j, a user consistently randomizes the pair $(\sigma_j, \mathsf{opk}_j)$ to obtain a randomized public key opk'_j. It then adapts the signature σ on its credential the same way, and presents opk'_j to the verifier. If the verifier accepts the signature σ_j on opk'_j (using vpk), it proceeds to verify σ using opk'_j. Issuer-policies can be specified by the entity that created the smart contract and defined within using the entity's verification key pair. Unlike the first approach where users choose the issuer's anonymity set, here it is determined by the policy maker.

For both of the above approaches we observe that the mercurial signature used in this work only provides a weak form of issuer-hiding. Given a signature that has been adapted to verify under a randomized public key pk' in the equivalence class of pk, the owner of pk can recognize it. Thus, issuers can know which transactions belong to users from their organizations (but not to which particular user) and which ones don't by reading the non-interactive showing proof (it contains the issuer's randomized public key). However, we argue that in the permissioned blockchain setting this provides a fair trade-off as a minimum traceability level is important for compliance and auditability purposes.

Our construction. Compared to [11], we make use of a hash function to apply the (strong) Fiat-Shamir transform while adding the previously discussed auditability and revocation features. Therefore we implement the ZKPoK's as Schnorr proofs (unlike [11] which followed Remark 1 from [14]).

In Figure 1 we present the setup, key generation, revocation and auditing algorithms. The setup algorithm also takes a bound q' on the maximun number of revocated pseudonyms for the revocation accumulator. The revocation authority is responsible for running the Revoke algorithm and updating the accumulator.

$\underline{\mathsf{Setup}(1^\lambda, \mathsf{aux})\text{:}}$
$(q, q') \leftarrow \mathsf{aux}$; $\mathbf{pick}\ \mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p^*$; $Q \xleftarrow{\$} \mathbb{G}_1$; $(\mathsf{rev}_{\mathsf{pp}}, \mathsf{rev}_{\mathsf{td}}) \xleftarrow{\$} \mathsf{RevAcc.Setup}(1^\lambda, q')$
$(\mathsf{BG}, \mathsf{scds}_{\mathsf{pp}}, \mathsf{scds}_{\mathsf{td}}) \xleftarrow{\$} \mathsf{SCDS.Setup}(1^\lambda, q)$; $(\mathsf{sps}_{\mathsf{pp}}, \mathsf{sps}_{\mathsf{td}}) \xleftarrow{\$} \mathsf{SPS\text{-}EQ.ParGen}(1^\lambda; \mathsf{BG})$
$\mathbf{return}\ (\mathcal{H}, \mathsf{BG}, \mathsf{rev}_{\mathsf{pp}}, Q, \mathsf{scds}_{\mathsf{pp}}, \mathsf{sps}_{\mathsf{pp}})$
$\underline{\mathsf{TSetup}(1^\lambda, \mathsf{aux})\text{:}}$
$(q, q') \leftarrow \mathsf{aux}$; $\mathbf{pick}\ \mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p^*$; $Q \xleftarrow{\$} \mathbb{G}_1$; $(\mathsf{rev}_{\mathsf{pp}}, \mathsf{rev}_{\mathsf{td}}) \xleftarrow{\$} \mathsf{RevAcc.Setup}(1^\lambda, q')$
$(\mathsf{BG}, \mathsf{scds}_{\mathsf{pp}}, \mathsf{scds}_{\mathsf{td}}) \xleftarrow{\$} \mathsf{SCDS.Setup}(1^\lambda, q)$; $(\mathsf{sps}_{\mathsf{pp}}, \mathsf{sps}_{\mathsf{td}}) \xleftarrow{\$} \mathsf{SPS\text{-}EQ.ParGen}(1^\lambda; \mathsf{BG})$
$\mathsf{td} = (\mathsf{rev}_{\mathsf{td}}, \mathsf{scds}_{\mathsf{td}}, \mathsf{sps}_{\mathsf{td}})$; $\mathbf{return}\ (\mathcal{H}, \mathsf{BG}, \mathsf{rev}_{\mathsf{pp}}, Q, \mathsf{scds}_{\mathsf{pp}}, \mathsf{sps}_{\mathsf{pp}}, \mathsf{td})$
$\underline{\mathsf{RevAcc.Setup}(1^\lambda, 1^q)\text{:}}$ $\mathsf{BG} \xleftarrow{\$} \mathsf{BGGen}(1^\lambda)$; $b \xleftarrow{\$} \mathbb{Z}_p^*$; $\mathbf{return}\ (\mathsf{BG}, (b^i P_1, b^i P_2)_{i \in [q]})$
$\underline{\mathsf{AAKGen}(\mathsf{pp})\text{:}}$ $\mathsf{ask} \xleftarrow{\$} Z_p^*$; $\mathsf{apk} \leftarrow \mathsf{ask} P_1$; $\mathbf{return}\ (\mathsf{ask}, \mathsf{apk})$
$\underline{\mathsf{RAKGen}(\mathsf{pp})\text{:}}$ $\mathsf{rsk} \xleftarrow{\$} Z_p^*$; $\mathsf{rpk} \leftarrow \mathsf{rsk} P_2$; $\mathbf{return}\ (\mathsf{rpk}, \mathsf{rsk})$
$\underline{\mathsf{OKGen}(\mathsf{pp})\text{:}}$ $\mathbf{return}\ \mathsf{SPS\text{-}EQ.KGen}(\mathsf{BG}, \mathsf{sps}_{\mathsf{pp}}, 3)$
$\underline{\mathsf{UKGen}(\mathsf{pp})\text{:}}$ $\mathsf{usk}_1, \mathsf{usk}_2 \xleftarrow{\$} Z_p^*$; $(\mathsf{upk}_1, \mathsf{upk}_2) \leftarrow (\mathsf{usk}_1 P_1, \mathsf{usk}_2 P_1)$
$\mathbf{return}\ ((\mathsf{usk}_1, \mathsf{usk}_2), (\mathsf{upk}_1, \mathsf{upk}_2))$
$\underline{\mathsf{RSetup}(\mathsf{pp}, (\mathsf{rsk}, \mathsf{rpk}), \mathsf{NYM}, \mathsf{RNYM})\text{:}}$
$(\mathit{\Pi}_{\mathsf{rev}}, \mathsf{aux}_{\mathsf{rev}}) \leftarrow \mathsf{RevAcc.Commit}(\mathsf{rev}_{\mathsf{pp}}, \mathsf{RNYM})$
$\mathbf{foreach}\ \mathsf{nym} \in \mathsf{NYM}\ \mathbf{do}\ \mathsf{WIT}[\mathsf{nym}] \leftarrow \mathsf{RevAcc.NonMemWit}(\mathsf{pp}, \mathit{\Pi}_{\mathsf{rev}}, \mathsf{aux}_{\mathsf{rev}}, \mathsf{nym})$
$\mathbf{return}\ ((\mathit{\Pi}_{\mathsf{rev}}, \mathsf{WIT}), \mathsf{aux}_{\mathsf{rev}})$
$\underline{\mathsf{RevAcc.Commit}(\mathsf{pp}, \mathcal{X}; \mathsf{rsk})\text{:}}$
$\mathbf{check}\ |\mathcal{X}| \leq q \wedge \nexists\, b' \in \mathcal{X} : b' P_1 = b P_1$; $\mathit{\Pi}_{\mathsf{rev}} \leftarrow \mathsf{rsk}^{-1} \cdot \mathsf{Ch}_{\mathcal{X}}(s) P_1$; $\mathsf{aux}_{\mathsf{rev}} \leftarrow \mathcal{X}$
$\mathbf{return}\ (\mathit{\Pi}_{\mathsf{rev}}, \mathsf{aux}_{\mathsf{rev}})$
$\underline{\mathsf{Revoke}(\mathsf{pp}, \mathbb{R}, \mathsf{aux}_{\mathsf{rev}}, \mathsf{nym}, b)\text{:}}$
$\mathbf{parse}\ \mathbb{R} = (\mathit{\Pi}_{\mathsf{rev}}, \mathsf{WIT})$; $\mathbf{parse}\ \mathsf{aux}_{\mathsf{rev}} = \mathsf{RNYM}$
$\mathbf{if}\ b = 1$
  $\mathsf{NYM} \leftarrow \mathsf{NYM} \setminus \{\mathsf{nym}\}$; $\mathsf{RNYM} \leftarrow \mathsf{RNYM} \cup \{\mathsf{nym}\}$
  $(\mathit{\Pi}'_{\mathsf{rev}}, \mathsf{aux}'_{\mathsf{rev}}) \leftarrow \mathsf{RevAcc.Add}(\mathsf{pp}, \mathit{\Pi}_{\mathsf{rev}}, \mathsf{RNYM}, \mathsf{nym})$
$\mathbf{else}$
  $\mathsf{NYM} \leftarrow \mathsf{NYM} \cup \{\mathsf{nym}\}$; $\mathsf{RNYM} \leftarrow \mathsf{RNYM} \setminus \{\mathsf{nym}\}$
  $(\mathit{\Pi}'_{\mathsf{rev}}, \mathsf{aux}'_{\mathsf{rev}}) \leftarrow \mathsf{RevAcc.Del}(\mathsf{pp}, \mathit{\Pi}_{\mathsf{rev}}, \mathsf{RNYM}, \mathsf{nym})$
$\mathbf{foreach}\ \mathsf{nym}' \in \mathsf{NYM}\ \mathbf{do}\ \mathsf{WIT}[\mathsf{nym}'] \leftarrow \mathsf{RevAcc.NonMemWit}(\mathsf{pp}, \mathit{\Pi}'_{\mathsf{rev}}, \mathsf{aux}'_{\mathsf{rev}}, \mathsf{nym}')$
$\mathbf{return}\ ((\mathit{\Pi}'_{\mathsf{rev}}, \mathsf{WIT}), \mathsf{aux}'_{\mathsf{rev}})$
$\underline{\mathsf{RevAcc.Add}(\mathsf{pp}, \mathsf{rsk}, \mathit{\Pi}_{\mathsf{rev}}, \mathsf{aux}_{\mathsf{rev}}, \mathsf{nym})\text{:}}$
$\mathbf{parse}\ \mathsf{aux}_{\mathsf{rev}} = \mathcal{X}$; $\mathcal{X} \leftarrow \mathcal{X} \cup \{\mathsf{nym}\}$; $\mathbf{return}\ \mathsf{RevAcc.Commit}(\mathsf{pp}, \mathcal{X}; \mathsf{rsk})$
$\underline{\mathsf{RevAcc.Del}(\mathsf{pp}, \mathsf{rsk}, \mathit{\Pi}_{\mathsf{rev}} \mathsf{aux}_{\mathsf{rev}}, \mathsf{nym})\text{:}}$
$\mathbf{parse}\ \mathsf{aux}_{\mathsf{rev}} = \mathcal{X}$; $\mathcal{X} \leftarrow \mathcal{X} \setminus \{\mathsf{nym}\}$; $\mathbf{return}\ \mathsf{RevAcc.Commit}(\mathsf{pp}, \mathcal{X}; \mathsf{rsk})$
$\underline{\mathsf{RevAcc.NonMemWit}(\mathsf{pp}, \mathit{\Pi}_{\mathsf{rev}}, \mathsf{aux}_{\mathsf{rev}}, \mathsf{nym})\text{:}}$
$\mathcal{X} \leftarrow \mathsf{aux}_{\mathsf{rev}}$; $\mathbf{check}\ \mathsf{nym} \notin \mathcal{X}$; Let $q(X)$ and $d \in \mathbb{Z}_p^*$ s.t. $\mathsf{Ch}_{\mathcal{X}}(X) = q(X)(X + \mathsf{nym}) + d$
$\mathbf{return}\ (q(b) P_2, d)$
$\underline{\mathsf{RevAcc.VerifyWit}(\mathsf{pp}, \mathit{\Pi}_{\mathsf{rev}}, \mathsf{nym}, \mathsf{wit}_{\mathsf{rev}})\text{:}}$
$(\mathsf{wit}^1_{\mathsf{rev}}, \mathsf{wit}^2_{\mathsf{rev}}) \leftarrow \mathsf{wit}_{\mathsf{rev}}$; $\mathbf{return}\ e(\mathit{\Pi}_{\mathsf{rev}}, \mathsf{rpk}) = e((b + \mathsf{nym}) P_1, \mathsf{wit}^1_{\mathsf{rev}}) e(\mathsf{wit}^2_{\mathsf{rev}} P_1, P_2)$
$\underline{\mathsf{AuditEnc}(\mathsf{upk}, \mathsf{apk})\text{:}}$ $\alpha \leftarrow \mathbb{Z}_p$; $\mathsf{enc} \leftarrow (\mathsf{upk} + \alpha \mathsf{apk}, \alpha P_1)$; $\mathbf{return}\ (\mathsf{enc}, \alpha)$
$\underline{\mathsf{AuditDec}(\mathsf{enc}, \mathsf{ask})\text{:}}$ $(\mathsf{enc}_1, \mathsf{enc}_2) \leftarrow \mathsf{enc}$; $\mathbf{return}\ (\mathsf{enc}_1 - \mathsf{ask} \cdot \mathsf{enc}_2)$
$\underline{\mathsf{AuditPrv}(\mathsf{enc}, \alpha, \mathsf{usk}, \mathsf{apk})\text{:}}$
$r_1, r_2 \leftarrow \mathbb{Z}_p$; $\mathsf{com}_1 \leftarrow r_1 P_1 + r_2 \mathsf{apk}$; $\mathsf{com}_2 \leftarrow r_2 P_1$; $c \leftarrow \mathcal{H}(\mathsf{com}_1, \mathsf{com}_2, \mathsf{enc})$
$z_1 \leftarrow r_1 + c \cdot \mathsf{usk}$; $z_2 \leftarrow r_2 + c \cdot \alpha$; $\mathbf{return}\ (c, z_1, z_2)$
$\underline{\mathsf{AuditVerify}(\mathsf{apk}, c, \mathsf{enc}, z_1, z_2)\text{:}}$
$\mathsf{com}_1 \leftarrow z_1 P_1 + z_2 \mathsf{apk} - c \mathsf{enc}_1$; $\mathsf{com}_2 \leftarrow z_2 P_1 - c \mathsf{enc}_2$; $c' \leftarrow \mathcal{H}(\mathsf{com}_1, \mathsf{com}_2, \mathsf{enc})$
$\mathbf{return}\ c' = c$

Fig. 1: Protego: setup, key generation, revocation and auditing algorithms.

$\underline{\mathsf{Obtain}(\mathsf{pp}, \mathsf{usk}, \mathsf{opk}, \mathsf{apk}, \mathcal{X}, \mathsf{nym})}$		$\underline{\mathsf{Issue}(\mathsf{pp}, \mathsf{upk}, \mathsf{osk}, \mathsf{apk}, \mathcal{X}, \mathsf{nym})}$
$r_1, r_2, r_3 \xleftarrow{\$} \mathbb{Z}_p^*;\ a_1 \leftarrow r_1 P_1;\ a_2 \leftarrow r_2 P_1$		
$a_3 \leftarrow r_3 Q;\ C_4 \leftarrow \mathsf{usk}_2(b + \mathsf{nym}) P_1$		
$C_5 \leftarrow \mathsf{usk}_2 Q;\ e \leftarrow \mathcal{H}(\mathsf{upk}_1, \mathsf{upk}_2, C_5, a_1, a_2, a_3)$		
$z_1 \leftarrow r_1 + e \cdot \mathsf{usk}_1$		
$z_2 \leftarrow r_2 + e \cdot \mathsf{usk}_2;\ z_3 \leftarrow r_3 + e \cdot \mathsf{usk}_2$		
$(C_1, O) \leftarrow \mathsf{SCDS.Commit}(\mathsf{scds}_{\mathsf{pp}}, \mathcal{X}; \mathsf{usk}_1)$		
$r_4 \xleftarrow{\$} \mathbb{Z}_p^*;\ C_2 \leftarrow r_4 \cdot C_1$		
$\Sigma \leftarrow (C_1, C_2, C_4, C_5, (a_i, z_i)_{i \in [3]})$	$\xrightarrow{\Sigma}$	$e \leftarrow \mathcal{H}(\mathsf{upk}_1, \mathsf{upk}_2, C_5, a_1, a_2, a_3)$
		check
		$z_1 P_1 = a_1 + e \cdot \mathsf{upk}_1$
		$z_2 P_1 = a_2 + e \cdot \mathsf{upk}_2;\ z_3 Q = a_3 + e \cdot C_5$
		$e(C_1, P_2) \neq e(\mathsf{upk}_1, \mathsf{Ch}_{\mathcal{X}}(s) P_2)$
		$\forall\, x \in \mathcal{X} : x P_1 \neq \mathsf{ek}_1^0$
check		$e(C_4, P_2) = e(\mathsf{upk}_2, (b + \mathsf{nym}) P_2)$
$\mathsf{SPS\text{-}EQ.Verify}(\mathsf{sps}_{\mathsf{pp}},$	$\xleftarrow{\sigma}$	$\sigma \leftarrow \mathsf{SPS\text{-}EQ.Sign}(\mathsf{sps}_{\mathsf{pp}},$
$(C_1, C_2, P_1, C_4, C_5, \mathsf{upk}_1, \mathsf{apk}), \sigma, \mathsf{opk})$		$(C_1, C_2, P_1, C_4, C_5, \mathsf{upk}_1, \mathsf{apk}), \mathsf{osk})$
return $\mathsf{cred} \leftarrow (C_1, C_4, C_5, \sigma, r_4, \mathsf{nym}, O)$		

Fig. 2: Protego: obtain and issue algorithms.

Obtain and Issue have constant-size communication and are given in Figure 2. For Show and Verify we present *Protego* and *Protego Duo*, depending on the issuer-hiding approach. Protego is given in Figure 3 and produces a variable-length proof as it relies on the (mallable) NIZK proof. Protego Duo produces a constant-size proof and is depicted in Figure 4. The differences are highlighted with grey. For both, after the credential is updated, the user randomizes the revocation accumulator, witnesses, and generates the Schnorr proofs. Following the auditing proof, the Fiat-Shamir transform is applied, the ZKPoK's and PoE's are computed, returning the showing proof. Verify takes a proof (depending on the case), computes the challenge and verifies each of the statements.

Security Model. Security of Protego relies on the properties from [14, 11, 13]. (1) *Correctness*, a credential showing with respect to a non-empty sets $\mathcal{S}$ and $\mathcal{D}$ of attributes always verify if it was issued honestly on some attribute set $\mathcal{X}$ with $\mathcal{S} \subset \mathcal{X}$ and $\mathcal{D} \not\subseteq \mathcal{X}$. (2) *Unforgeablility*, given at least one non-empty set $\mathcal{S} \subset \mathcal{X}$ or $\mathcal{D} \not\subseteq \mathcal{X}$, a user in possession of a credential for the attribute set $\mathcal{X}$ cannot perform a valid showing for $\mathcal{D} \subset \mathcal{X}$ nor for $\mathcal{S} \not\subseteq \mathcal{X}$. Moreover, revocated users cannot perform valid showings and no coalition of malicious users can combine their credentials and prove possession of a set of attributes which no single member has. This holds even after seeing showings of arbitrary credentials by honest users. (3) *Anonymity*, during a showing, no verifier and no (malicious) organization (even if they collude) is able to identify the user or learn anything about the user, except that she owns a valid credential for the shown attributes. Furthermore, different showings of the same credential are unlinkable.

The formal security model, adapted from [11] and [13] to consider non-interactive showings, auditability and issuer-hiding features, is given in Appendix C. We do not consider replay-attacks as in the previous models since for

$\underline{\mathsf{Show}(\mathsf{pp}, \mathsf{usk}, \mathsf{upk}, \mathsf{opk}_j, \mathsf{cred}, \mathcal{S}, \mathcal{D}, (\mathsf{opk}_i)_{i\in[n]}, (\mathsf{opk}_j^i, w_j^i)_{i\in[2]}, \Omega, \mathbb{R}, \mathsf{apk}, \mathsf{tx})}$

$(C_1, C_4, C_5, \sigma, r, \mathsf{nym}, O) \leftarrow \mathsf{cred};\ (\Pi_{\mathsf{rev}}, \mathsf{WIT}) \leftarrow \mathbb{R}; \beta, \mu, \rho, \gamma, \tau, (r_i)_{i\in[5]} \xleftarrow{\$} \mathbb{Z}_p^*$
if $O = (1, (o_1, o_2))$ **then** $O' = (1, (\mu \cdot o_1, o_2))$ **else** $O' = \mu \cdot O$
$\mathsf{opk}_j' \leftarrow \mathsf{ConvertPK}(\mathsf{opk}_j, \rho w_j^1 + \gamma w_j^2); \Omega' \leftarrow \mathsf{SH.ZKEval}(\mathsf{opk}_j^1, \mathsf{opk}_j^2, \Omega; \rho, \gamma)$
$\sigma' \xleftarrow{\$} \mathsf{SPS\text{-}EQ.ChgRep}(\mathsf{sps}_{\mathsf{pp}}, (C_1, rC_1, P_1, C_4, C_5, \mathsf{upk}_1, \mathsf{apk}), \sigma, \mu, \rho w_j^1 + \gamma w_j^2, \mathsf{opk}_j)$
$\mathsf{cred}' \leftarrow ((C_i)_{i\in[7]} = \mu \cdot (C_1, rC_1, P_1, C_4, C_5, \mathsf{upk}_1, \mathsf{apk}), \sigma')$
$\mathsf{wit} \leftarrow \mathsf{SCDS.OpenSS}(\mathsf{scds}_{\mathsf{pp}}, C_1, \mathcal{S}, O'); \underline{\mathsf{wit}} \leftarrow \mathsf{SCDS.OpenDS}(\mathsf{scds}_{\mathsf{pp}}, C_1, \mathcal{D}, O')$
$\mathsf{wit}_{\mathsf{rev}} \leftarrow \mathsf{WIT}[\mathsf{nym}]; \mathsf{wit}_{\mathsf{rev}}' \leftarrow (\tau \mathsf{wit}_{\mathsf{rev}}^1, \mathsf{usk}_2 \mu\tau \mathsf{wit}_{\mathsf{rev}}^2 P_1)$
$a_1 \leftarrow r_1 C_1; a_2 \leftarrow r_2 P_1; a_3 \leftarrow r_3 \Pi_{\mathsf{rev}}; a_4 \leftarrow r_4 Q; a_5 \leftarrow r_5 P_1; \Pi_{\mathsf{rev}}' \leftarrow (\mathsf{usk}_2 \mu\tau)\Pi_{\mathsf{rev}}$
$(\mathsf{enc}, \alpha) \leftarrow \mathsf{AuditEnc}(\mathsf{apk}, \mathsf{upk}_1); t_1 = \beta P_2; t_2 = \beta\mu P_2; t_3 = \alpha\beta P_2$
$\Pi \leftarrow \mathsf{AuditPrv}(\mathsf{enc}, \alpha, \mathsf{usk}, \mathsf{apk})$
$e \leftarrow \mathcal{H}(\mathcal{S}, \mathcal{D}, \mathsf{apk}, \mathsf{tx}, \mathsf{enc}, \Pi, \mathsf{opk}_j', (\mathsf{opk}_i)_{i\in[n]}, \Omega', (a_i)_{i\in[5]}, (t_i)_{i\in[3]}, \mathsf{cred}', \mathsf{wit}, \underline{\mathsf{wit}}, \mathsf{tx},$
$C_2, C_3, \Pi_{\mathsf{rev}}', C_5, \mathsf{wit}_{\mathsf{rev}}')$
$z_1 \leftarrow r_1 + e \cdot r;\ z_2 \leftarrow r_2 + e \cdot \mu;\ z_3 \leftarrow r_3 + e \cdot (\mathsf{usk}_2 \mu\tau);\ z_4 \leftarrow r_4 + e \cdot (\mathsf{usk}_2 \mu)$
$z_5 \leftarrow r_5 + e \cdot (\mathsf{usk}_2 \mu\tau \mathsf{wit}_{\mathsf{rev}}^2)$
$\pi_1 \leftarrow \mathsf{SCDS.PoE}(\mathsf{scds}_{\mathsf{pp}}, \mathcal{S}, e); \pi_2 \leftarrow \mathsf{SCDS.PoE}(\mathsf{scds}_{\mathsf{pp}}, \mathcal{D}, e)$
return $(\mathsf{enc}, (t_i)_{i\in[3]}, \mathsf{opk}', (\mathsf{opk}_i)_{i\in[n]}, \Omega', \mathsf{cred}', \mathsf{wit}, \underline{\mathsf{wit}}, \mathsf{wit}_{\mathsf{rev}}', \Pi_{\mathsf{rev}}', \Pi, \pi_1, \pi_2, (a_i, z_i)_{i\in[5]})$

$\underline{\mathsf{Verify}(\mathsf{pp}, \mathcal{S}, \mathcal{D}, \Pi_{\mathsf{rev}}, \mathsf{rpk}, \mathsf{apk}, \mathsf{tx}, \Omega)}$

$(\mathsf{enc}, (t_i)_{i\in[3]}, \mathsf{opk}', (\mathsf{opk}_i)_{i\in[n]}, \Omega', \mathsf{cred}', \mathsf{wit}, \underline{\mathsf{wit}}, \mathsf{wit}_{\mathsf{rev}}', \Pi_{\mathsf{rev}}', \Pi, \pi_1, \pi_2, (a_i, z_i)_{i\in[5]}) \leftarrow \Omega$
$(C_1, C_2, C_3, C_4, C_5, C_6, C_7, \sigma) \leftarrow \mathsf{cred}'$
$e \leftarrow \mathcal{H}(\mathcal{S}, \mathcal{D}, \mathsf{apk}, \mathsf{tx}, \mathsf{enc}, \Pi, \mathsf{opk}', (\mathsf{opk}_i)_{i\in[n]}, \Omega', (a_i)_{i\in[5]}, (t_i)_{i\in[3]}, \mathsf{cred}', \mathsf{wit}, \underline{\mathsf{wit}}, \mathsf{tx},$
$C_2, C_3, \Pi_{\mathsf{rev}}', C_5, \mathsf{wit}_{\mathsf{rev}}')$
check
$z_1 C_1 = a_1 + eC_2;\ z_2 P_1 = a_2 + eC_3;\ z_3 \Pi_{\mathsf{rev}} = a_3 + e\Pi_{\mathsf{rev}}';\ z_4 Q = a_4 + eC_5$
$z_5 P_1 = a_5 + e\mathsf{wit}_{\mathsf{rev}}';\ \mathsf{RevAcc.VerifyWit}(\Pi_{\mathsf{rev}}', C_4, \mathsf{wit}_{\mathsf{rev}}');\ \mathsf{AuditVerify}(\mathsf{enc}, \Pi_2)$
$e(\mathsf{enc}_1, t_2) = e(C_6, t_1) + e(C_7, t_3);\ e(\mathsf{enc}_2, t_2) = e(C_3, t_3);\ e(\mathsf{enc}_2, t_1) = e(P_1, t_3)$
$\mathsf{SCDS.VerifySS}(C_1, \mathcal{S}, \mathsf{wit}; \pi_1, e);\ \mathsf{SCDS.VerifyDS}(C_1, \mathcal{D}, \underline{\mathsf{wit}}; \pi_2, e)$
$\mathsf{SH.Verify}((\mathsf{opk}_i)_{i\in[n]}, \mathsf{opk}', \Omega');\ \mathsf{SPS\text{-}EQ.Verify}(\mathsf{cred}', \mathsf{opk}')$

Fig. 3: Protego: show and verify algorithms.

the same transaction they can be trivially detected. Next, we present the main theorems and proofs for Protego (which are analogous for Protego Duo). The rest is referred to the extended version of this work (also under Appendix C).

Theorem 1. *If the* q-co-DL *assumption holds, the ZKPoK's have perfect ZK,* SCDS *is sound,* SPS-EQ *is* EUF-CMA *secure, and* RevAcc *is collision-free then Protego is unforgeable.*

Proof Sketch. The proof follows from [11] (Th. 6) and [13] (Th. 3) whereby we assume there is an efficient adversary $\mathcal{A}$ winning the unforgeability game with non-negligible probability. We use $\mathcal{A}$ considering the following types of attacks:

Type 1. Adversary $\mathcal{A}$ conducts a valid showing so that $\mathsf{nym}^* = \perp$. Then we construct an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to break the EUF-CMA security.

Type 2. Adversary $\mathcal{A}$ manages to conduct a showing accepted by the verifier using the credential of user i^* under nym^* with respect to $\mathcal{S}^*$ such that $\mathcal{S}^* \not\subseteq \mathtt{ATTR}[\mathsf{nym}]$ or with respect to $\mathcal{D}^*$ such that $\mathcal{D}^* \subseteq \mathtt{ATTR}[\mathsf{nym}]$ holds. Then we construct an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to break the soundness of the set-commitment scheme SCDS.

$\underline{\mathsf{Show}(\mathsf{pp}, \mathsf{usk}, \mathsf{upk}, \mathsf{opk}_j, \mathsf{cred}, \mathcal{S}, \mathcal{D}, \mathsf{opk}_j, \sigma_j, \mathbb{R}, \mathsf{apk}, \mathsf{tx})}$

$(C_1, C_4, C_5, \sigma, r, \mathsf{nym}, O) \leftarrow \mathsf{cred};\ (\Pi_{\mathsf{rev}}, \mathsf{WIT}) \leftarrow \mathbb{R}; \beta, \mu, \rho, \gamma, \tau, (r_i)_{i\in[5]} \xleftarrow{\$} \mathbb{Z}_p^*$

if $O = (1, (o_1, o_2))$ **then** $O' = (1, (\mu \cdot o_1, o_2))$ **else** $O' = \mu \cdot O$

$\mathsf{opk}'_j \leftarrow \mathsf{ConvertPK}(\mathsf{opk}_j, \rho); \sigma'_j \xleftarrow{\$} \mathsf{SPS\text{-}EQ.ChgRep}(\mathsf{sps}_{\mathsf{pp}}, \mathsf{opk}_j, \sigma_j, \rho)$

$\sigma' \xleftarrow{\$} \mathsf{SPS\text{-}EQ.ChgRep}(\mathsf{sps}_{\mathsf{pp}}, (C_1, rC_1, P_1, C_4, C_5, \mathsf{upk}_1, \mathsf{apk}), \sigma, \mu, \rho, \mathsf{opk}_j)$

$\mathsf{cred}' \leftarrow ((C_i)_{i\in[7]} = \mu \cdot (C_1, rC_1, P_1, C_4, C_5, \mathsf{upk}_1, \mathsf{apk}), \sigma')$

$\mathsf{wit} \leftarrow \mathsf{SCDS.OpenSS}(\mathsf{scds}_{\mathsf{pp}}, C_1, \mathcal{S}, O'); \underline{\mathsf{wit}} \leftarrow \mathsf{SCDS.OpenDS}(\mathsf{scds}_{\mathsf{pp}}, C_1, \mathcal{D}, O')$

$\mathsf{wit}_{\mathsf{rev}} \leftarrow \mathsf{WIT}[\mathsf{nym}]; \mathsf{wit}'_{\mathsf{rev}} \leftarrow (\tau \mathsf{wit}^1_{\mathsf{rev}}, \mathsf{usk}_2 \mu\tau \mathsf{wit}^2_{\mathsf{rev}} P_1)$

$a_1 \leftarrow r_1 C_1; a_2 \leftarrow r_2 P_1; a_3 \leftarrow r_3 \Pi_{\mathsf{rev}}; a_4 \leftarrow r_4 Q; a_5 \leftarrow r_5 P_1; \Pi'_{\mathsf{rev}} \leftarrow (\mathsf{usk}_2 \mu\tau)\Pi_{\mathsf{rev}}$

$(\mathsf{enc}, \alpha) \leftarrow \mathsf{AuditEnc}(\mathsf{apk}, \mathsf{upk}_1); t_1 = \beta P_2; t_2 = \beta\mu P_2; t_3 = \alpha\beta P_2$

$\Pi \leftarrow \mathsf{AuditPrv}(\mathsf{enc}, \alpha, \mathsf{usk}, \mathsf{apk})$

$e \leftarrow \mathcal{H}(\mathcal{S}, \mathcal{D}, \mathsf{apk}, \mathsf{tx}, \mathsf{enc}, \Pi, \mathsf{opk}'_j, \sigma'_j, (a_i)_{i\in[5]}, (t_i)_{i\in[3]}, \mathsf{cred}', \mathsf{wit}, \underline{\mathsf{wit}}, \mathsf{tx},$
$C_2, C_3, \Pi'_{\mathsf{rev}}, C_5, \mathsf{wit}'_{\mathsf{rev}})$

$z_1 \leftarrow r_1 + e \cdot r;\ z_2 \leftarrow r_2 + e \cdot \mu;\ z_3 \leftarrow r_3 + e \cdot (\mathsf{usk}_2 \mu\tau);\ z_4 \leftarrow r_4 + e \cdot (\mathsf{usk}_2 \mu)$

$z_5 \leftarrow r_5 + e \cdot (\mathsf{usk}_2 \mu\tau \mathsf{wit}^2_{\mathsf{rev}})$

$\pi_1 \leftarrow \mathsf{SCDS.PoE}(\mathsf{scds}_{\mathsf{pp}}, \mathcal{S}, e); \pi_2 \leftarrow \mathsf{SCDS.PoE}(\mathsf{scds}_{\mathsf{pp}}, \mathcal{D}, e)$

return $(\mathsf{enc}, (t_i)_{i\in[3]}, \mathsf{opk}'_j, \sigma'_j, \mathsf{cred}', \mathsf{wit}, \underline{\mathsf{wit}}, \mathsf{wit}'_{\mathsf{rev}}, \Pi'_{\mathsf{rev}}, \Pi, \pi_1, \pi_2, (a_i, z_i)_{i\in[5]})$

$\underline{\mathsf{Verify}(\mathsf{pp}, \mathcal{S}, \mathcal{D}, \Pi_{\mathsf{rev}}, \mathsf{rpk}, \mathsf{apk}, \mathsf{vpk}, \mathsf{tx}, \Omega)}$

$(\mathsf{enc}, (t_i)_{i\in[3]}, \mathsf{opk}', \sigma', \mathsf{cred}', \mathsf{wit}, \underline{\mathsf{wit}}, \mathsf{wit}'_{\mathsf{rev}}, \Pi'_{\mathsf{rev}}, \Pi, \pi_1, \pi_2, (a_i, z_i)_{i\in[5]}) \leftarrow \Omega$

$(C_1, C_2, C_3, C_4, C_5, C_6, C_7, \sigma) \leftarrow \mathsf{cred}'$

$e \leftarrow \mathcal{H}(\mathcal{S}, \mathcal{D}, \mathsf{apk}, \mathsf{tx}, \mathsf{enc}, \Pi, \mathsf{opk}', \sigma', (a_i)_{i\in[5]}, (t_i)_{i\in[3]}, \mathsf{cred}', \mathsf{wit}, \underline{\mathsf{wit}}, \mathsf{tx},$
$C_2, C_3, \Pi'_{\mathsf{rev}}, C_5, \mathsf{wit}'_{\mathsf{rev}})$

check

$z_1 C_1 = a_1 + eC_2;\ z_2 P_1 = a_2 + eC_3;\ z_3 \Pi_{\mathsf{rev}} = a_3 + e\Pi'_{\mathsf{rev}};\ z_4 Q = a_4 + eC_5$

$z_5 P_1 = a_5 + e\mathsf{wit}'_{\mathsf{rev}};\ \mathsf{RevAcc.VerifyWit}(\Pi'_{\mathsf{rev}}, C_4, \mathsf{wit}'_{\mathsf{rev}});\ \mathsf{AuditVerify}(\mathsf{enc}, \Pi_2)$

$e(\mathsf{enc}_1, t_2) = e(C_6, t_1) + e(C_7, t_3);\ e(\mathsf{enc}_2, t_2) = e(C_3, t_3);\ e(\mathsf{enc}_2, t_1) = e(P_1, t_3)$

$\mathsf{SCDS.VerifySS}(C_1, \mathcal{S}, \mathsf{wit}; \pi_1, e);\ \mathsf{SCDS.VerifyDS}(C_1, \mathcal{D}, \underline{\mathsf{wit}}; \pi_2, e)$

$\mathsf{SPS\text{-}EQ.Verify}(\mathsf{opk}', \mathsf{vpk});\ \mathsf{SPS\text{-}EQ.Verify}(\mathsf{cred}', \mathsf{opk}')$

Fig. 4: Protego Duo: show and verify algorithms.

Type 3. Adversary $\mathcal{A}$ manages to conduct a showing accepted by the verifier reusing a showing based on the credential of a user i^* under nym^* with $i^* \in \mathtt{HU}$, whose secret usk_{i^*} and credentials it does not know.

Type 4. Adversary $\mathcal{A}$ manages to conduct a showing accepted by the verifier using some credential corresponding to a revoked pseudonym $\mathsf{nym}^* \in \mathsf{RNYM}$. Then, we construct an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to break the binding property of the revocation accumulator RevAcc.

Types 1 and 2 follow the proofs of [11] (Th. 6) as the underlying primitives remain unchanged. For Type 3, we leverage the fact that reusing a showing would only allow the adversary to generate a valid showing for *the same* original transaction tx (that is timestamped), and hence, we do not consider it as an attack. Observe that any modification done to the original tx will lead to a different challenge and thus the rest of the proofs (showing, revocation and auditing) will not pass. Finally, Type 4 follows from [13] (Th. 3). □

Theorem 2. *If the DDH assumption holds, the* SPS-EQ *perfectly adapts signatures, and* $\mathcal{H}$ *is assumed to be a random oracle, then Protego is anonymous.*

Proof Sketch. The proof follows from [11] (Th. 7) and [13] (Th. 4). However, we must also to take into account the RO model and the addition of the auditing features. The extra credential components for the auditing are randomized during every credential showing like the rest of the components. Similarly, the user generates a new encryption of the auditor's public key with a fresh α, while a fresh β is used to randomize the values t_i. Since ElGamal encryption is IND-CPA secure and key-private [5], the ciphertexts produced by the user are indistinguishable and do not leak information about the user's public key nor the auditor's. □

Theorem 3. *If the algorithms* $\mathsf{AuditPrv}$ *and* $\mathsf{AuditVerify}$ *are a NIZK proof system and the* $\mathsf{SPS\text{-}EQ}$ *is* EUF-CMA *secure then Protego is auditable.*

Proof. If the verification returns true, we have that $\exists\ (\mathsf{enc}_1^*, \mathsf{enc}_2^*) = ((\delta^* + \alpha^*\mathsf{ask})P_1, \alpha^* P_1)$ for some δ^* and α^* chosen by the adversary. Moreover, because of the unforgeability of the signature scheme, the verification implies that $C_3 = \mu^* P_1$, $C_6 = \mu^*\mathsf{usk}_1 P_1$ and $C_7 = \mu^*\mathsf{ask}P_1$ for some μ^* chosen by the adversary. As a result, we can re-write the pairing equations for the audit proof as:

$$
\begin{aligned}
e(\alpha^* P_1, t_2^*) &= e(\mu^* P_1, t_3^*)\\
e(\alpha^* P_1, t_1^*) &= e(P_1, t_3^*)\\
e((\delta^* + \alpha^*\mathsf{ask})P_1, t_2^*) &= e(\mu^*\mathsf{usk}_1 P_1, t_1^*) + e(\mu^*\mathsf{ask}P_1, t_3^*)
\end{aligned}
$$

where t_1^*, t_2^* and t_3^* are also chosen by the adversary. We show that $\delta^* = \mathsf{usk}_1$, which implies that $\mathsf{upk}_1 = \mathsf{AuditDec}(\mathsf{enc}, \mathsf{ask})$. Looking at the first two equations in the target group, we have that $\alpha^* t_2^* = \mu^* t_3^*$ and $\alpha^* t_1^* = t_3^*$, concluding that $t_2^* = \mu^* t_1^*$. Replacing t_2^* and t_3^* in third one and simplyfing we obtain:

$$
\begin{aligned}
(\delta^* + \alpha^*\mathsf{ask})\mu^* t_1^* &= \mu^*\mathsf{usk}_1 t_1^* + \mu\mathsf{ask}\alpha^* t_1^*\\
\mu^*\delta^* t_1^* + \mu^*\alpha^*\mathsf{ask}t_1^* &= \mu^*\mathsf{usk}_1 t_1^* + \mu^*\alpha^*\mathsf{ask}t_1^*
\end{aligned}
$$

deducing that $\delta^* = \mathsf{usk}_1$. □

Integration with Fabric. A multi-party computation ceremony can be run for the CRS generation of the Setup algorithm to ensure that no organization knows the trapdoors of the different components. As we are in the permissioned setting it is plausible to assume that at least one of the organizations is honest.

By allowing *users and endorsers* to obtain credentials, both can produce showing proofs. Users can generate showing proofs to prove that they satisfy the access policy for the execution of a particular transaction proposal. Furthermore, by computing the PoE's, the verification time for endorsers improves substantially. Similarly, endorsers can prove that they satisfy a given endorsement policy attaching a showing proof to their endorements. Even if the endorsement policies are defined in a privacy-preserving way as suggested in [3], endorsers can still compute selective AND and NAND clauses for the respective pseudonyms defined by the policy using their credentials. Endorsers should also use the read and write sets to from the transaction proposals to generate their showing proofs.

4 Evaluation

We implemented a prototype version of Protego and Protego Duo in Rust using the bls12-381 curve and the BLAKE3 hash function. The source code and related documentation are provided in [1]. Our signature implementation is based on the one from [8] but using the bls12-381 curve instead of Barreto-Naehrig curves [4]. As a result, we obtain times up to 67% faster when compared to [8]. To run the benchmarks a regular laptop (i7-1165G7 CPU & 16GB RAM) was used with no extra optimizations, using the nightly compiler, and the *Criterion* library.

When issuing a credential for 10 attributes, the protocols Issue and Obtain take roughly 32 ms and 28 ms respectively. Both scale linearly on the number of attributes to be issued. To evaluate the showing and verification protocols we considered the PoE in the showing protocol. Therefore, the verification running time remains (almost) constant[5] regardless the number of shown attributes, credential size, and issuer-hiding approach. If the proof of exponentiation is disabled, the showing running time would be smaller while the verification one would increase linearly with the number of shown attributes.

Comparison with the Idemix extension from [7]. A showing proof in Protego Duo is constant-size, surpassing the latest work from [7] in which the proof size grows linearly with the number of *attributes and delegation levels*. Moreover, the computational cost for the prover and verifier also grows linearly with the number of attributes in the credential and delegation levels for [7]. In Protego Duo, the prover computational cost is $O(n-k)$ for showings involving k-attributes out of n, which in practice is much better. Verification cost in Protego and Protego Duo is almost constant (or $\mathrm{O}(k)$ if the PoE is disabled).

The two works are compared in Figure 5 using the same hardware. For [7], we consider a delegation level $L = 2$, which would correspond to a user level given that the root authority is at $L = 0$ and organizations start at $L = 1$. Regarding the attributes, [7] we could only retrieve information considering proofs for credential possesion below ten attributes (assuming a minimal overhead when all attributes are shown as authors suggest). Therefore, we report credential possesions for [7] considering up to 8 attributes, and selective disclosures of k-out-of-10 attributes in ours. For Protego, we consider five authorities for the NIZK proof, which would suffice for practical scenarios like a consortium of pharmaceuticals. Exact times for the values in Figure 5 are also given in the extended version (Table 1, Appendix E), where we also compare the main algorithms individually.

5 Conclusions & Future Work

We presented here the first SPS-EQ credential scheme modified to work with permissioned blockchains. The versatility of Protego alongside the efficiency gains (at least twice as fast as the most recent Idemix extension), enables a broader

[5] Asymptotic complexity is O(1) (considering exponentiations and pairings) but some multiplications depending on the shown attributes are required, hence the difference.

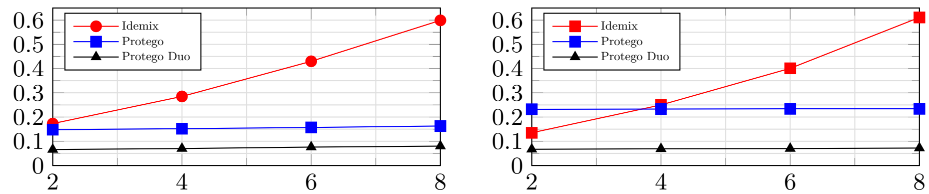

Fig. 5: From left to right, showing and verification times (in seconds) for the different schemes considering credential showings for 2, 4, 6 and 8 attributes.

scope of applications in such a setting. Depending on the context, the PoE's can be computed or not, the credential issuer can be hidden or not, and one can select only subsets or disjoint sets to generate the proofs. Similarly, auditability and revocation features can be considered as optional, showing its flexibility.

As future directions to explore, we consider the following points: (1) adding confidentiality of transactions to a Protego-like credential scheme, (2) adding more power to the users (*i.e.,* how to define precise notions of user-invoked regulatory measures), and (3) extend our results to the multi-authority setting, where users can get attributes from multiple authorities instead of a single one.

References

1. Implementation. Available upon request.
2. Elli Androulaki, Jan Camenisch, Angelo De Caro, Maria Dubovitskaya, Kaoutar Elkhiyaoui, and Björn Tackmann. Privacy-preserving auditable token payments in a permissioned blockchain system. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, AFT '20, page 255–267, New York, NY, USA, 2020. Association for Computing Machinery.
3. Elli Androulaki, Angelo De Caro, Matthias Neugschwandtner, and Alessandro Sorniotti. Endorsement in hyperledger fabric. In *2019 IEEE International Conference on Blockchain (Blockchain)*, pages 510–519, 2019.
4. Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *SAC 2005*, volume 3897 of *LNCS*, pages 319–331. Springer, Heidelberg, August 2006.
5. Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 566–582. Springer, Heidelberg, December 2001.
6. Jan Bobolz, Fabian Eidens, Stephan Krenn, Sebastian Ramacher, and Kai Samelin. Issuer-Hiding Attribute-Based Credentials. In Mauro Conti, Marc Stevens, and Stephan Krenn, editors, *Cryptology and Network Security*, pages 158–178, Cham, 2021. Springer International Publishing.
7. Dmytro Bogatov, Angelo De Caro, Kaoutar Elkhiyaoui, and Björn Tackmann. Anonymous transactions with revocation and auditing in hyperledger fabric. In Mauro Conti, Marc Stevens, and Stephan Krenn, editors, *Cryptology and Network Security*, pages 435–459, Cham, 2021. Springer International Publishing.
8. Michael Burkhart. Mercurial signatures implementation. Github, 2020. `https://github.com/burkh4rt/Mercurial-Signatures`.
9. Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer, Heidelberg, August 2004.

10. Jan Camenisch, Sebastian Mödersheim, and Dieter Sommer. A formal model of identity mixer. In *Proceedings of the 15th International Conference on Formal Methods for Industrial Critical Systems*, FMICS'10, page 198–214, Berlin, Heidelberg, 2010. Springer-Verlag.
11. Aisling Connolly, Pascal Lafourcade, and Octavio Perez Kempner. Improved Constructions of Anonymous Credentials from Structure-Preserving Signatures on Equivalence Classes. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *Public-Key Cryptography – PKC 2022*, pages 409–438, Cham, 2022. Springer International Publishing.
12. Elizabeth C. Crites and Anna Lysyanskaya. Delegatable anonymous credentials from mercurial signatures. In Mitsuru Matsui, editor, *CT-RSA 2019*, volume 11405 of *LNCS*, pages 535–555. Springer, Heidelberg, March 2019.
13. David Derler, Christian Hanser, and Daniel Slamanig. A new approach to efficient revocable attribute-based anonymous credentials. In Jens Groth, editor, *15th IMA International Conference on Cryptography and Coding*, volume 9496 of *LNCS*, pages 57–74. Springer, Heidelberg, December 2015.
14. Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. *Journal of Cryptology*, 32(2):498–546, April 2019.
15. Hui Kang, Ting Dai, Nerla Jean-Louis, Shu Tao, and Xiaohui Gu. Fabzk: Supporting privacy-preserving, auditable smart contracts in hyperledger fabric. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 543–555, 2019.
16. Subhra Mazumdar and Sushmita Ruj. Design of anonymous endorsement system in hyperledger fabric. *IEEE Transactions on Emerging Topics in Computing*, pages 1–1, 2019.
17. Neha Narula, Willy Vasquez, and Madars Virza. Zkledger: Privacy-preserving auditing for distributed ledgers. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation*, NSDI'18, page 65–80, USA, 2018. USENIX Association.
18. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990.
19. IBM Research Zurich. Specification of the identity mixer cryptographic library v2.3.0., 2013.

A Set-Commitment scheme supporting Disjoint Sets

Syntax. A Set-Commitment scheme supporting Disjoint Sets is a 9-tuple of algorithms. (1) $\mathsf{Setup}(1^\lambda, 1^q)$ takes as input a security parameter λ and an upper bound q for the cardinality of committed sets. It outputs public parameters pp, discarding the trapdoor key s. (2) $\mathsf{TSetup}(1^\lambda, 1^q)$ is like Setup but returns the trapdoor key. (3) $\mathsf{Commit}(\mathsf{pp}, \mathcal{X})$ takes as input pp and a set $\mathcal{X}$ with $1 \leq |\mathcal{X}| \leq q$. It outputs a commitment C on $\mathcal{X}$ and opening information O. (4) $\mathsf{Open}(\mathsf{pp}, C, \mathcal{X}, O)$ takes as input pp, a commitment C, a set $\mathcal{X}$, and opening information O. It outputs 1 if and only if O is a valid opening of C on $\mathcal{X}$. (5) $\mathsf{OpenSS}(\mathsf{pp}, C, \mathcal{X}, O, \mathcal{S})$ takes as input pp, a commitment C, a set $\mathcal{X}$, opening information O, and a non-empty set $\mathcal{S}$. If $\mathcal{S}$ is a subset of $\mathcal{X}$ committed to in C, it outputs a witness wit that attests to it. Otherwise, outputs $\perp$. (6) $\mathsf{OpenDS}(\mathsf{pp}, C, \mathcal{X}, O, \mathcal{D})$ takes as input pp, a commitment C, a set $\mathcal{X}$, opening information O, and a non-empty set $\mathcal{D}$. If $\mathcal{D}$ is disjoint from $\mathcal{X}$ committed to in C, it outputs a witness $\underline{\mathsf{wit}}$ that attests to it. Otherwise, outputs $\perp$. (7) $\mathsf{VerifySS}(\mathsf{pp}, C, \mathcal{S}, \mathsf{wit})$ takes as input pp, a commitment C, a non-empty set $\mathcal{S}$, and a witness wit. If wit is a valid witness for $\mathcal{S}$ a subset of the set committed to in C, it outputs 1 and otherwise $\perp$. (8) $\mathsf{VerifyDS}(\mathsf{pp}, C, \mathcal{D}, \underline{\mathsf{wit}})$ takes as input pp, a commitment C, a non-empty set $\mathcal{D}$, and a witness wit. If $\underline{\mathsf{wit}}$ is a valid witness for $\mathcal{D}$ being disjoint from the set committed to in C, it outputs 1 and otherwise $\perp$. (9) $\mathsf{PoE}(\mathsf{pp}, \mathcal{X}, \alpha)$ takes as input pp, a non-empty set $\mathcal{X}$, and a randomly-chosen value α. It computes a proof of exponentiation with respect to $\mathcal{X}$ and outputs a proof π_Q and a witness Q.

Security Properties. Correctness requires that (1) for a set $\mathcal{X}$, $\mathsf{Open}(\mathsf{Commit}(\mathsf{pp},\mathcal{X}),\mathcal{X}) = 1$, (2) for $\mathcal{S}$ a subset of $\mathcal{X}$, $\mathsf{VerifySS}(\mathcal{S}, \mathsf{OpenSS}(\mathsf{Commit}(\mathsf{pp}, \mathcal{X}))) = 1$, and (3) for all possible sets $\mathcal{D}$ disjoint from $\mathcal{X}$, $\mathsf{VerifyDS}(\mathcal{D}, \mathsf{OpenDS}(\mathsf{Commit}(\mathsf{pp}, \mathcal{X})))=1$. The scheme should also be (1) *binding* whereby each commitment overwhelmingly pertains to one particular set of attributes, (2) *hiding* whereby an adversary, given access to opening oracles, should not be able to distinguish which of two sets a commitment was generated on, and (3) *sound* in that sets which are not subsets of the committed set do not verify under $\mathsf{VerifySS}$, and sets that are not disjoint from the committed set do not verify under $\mathsf{VerifyDS}$.

B Structure-Preserving Signature on Equivalence Classes

Syntax. A SPS-EQ is a 5-tuple of algorithms. (1) $\mathsf{ParGen}(1^\lambda)$ takes as input a security parameter λ and returns public parameters pp with an asymmetric bilinear group BG. (2) $\mathsf{KGen}(\mathsf{pp}, \ell)$ takes as input pp, a message-length ℓ, and outputs a key pair $(\mathsf{sk}, \mathsf{pk})$. (3) $\mathsf{Sign}(\mathsf{sk}, m)$ takes as input sk and message m. It outputs a signature σ on m where $m \in (\mathbb{G}_i^*)^\ell$ is a representative for a class $[m]_\mathcal{R}$. (4) $\mathsf{ChgRep}(m, \sigma, \mu, \mathsf{pk})$ takes as input m, σ, μ, pk. It deterministically computes an updated signature σ' on new representative $m^* = \mu m$ and returns (m^*, σ'). (5) $\mathsf{Verify}(m, \sigma, \mathsf{pk})]$ takes m, σ and pk, and outputs 1 iff σ is valid for m.

Security Properties. Correctness requires for all values above that (1) $\mathsf{Verify}(m, \mathsf{Sign}(\mathsf{sk}, m), \mathsf{pk}) = 1$ and (2) that $\mathsf{Verify}(\mathsf{ChgRep}(m, \mathsf{Sign}(\mathsf{sk}, m), \mu, \mathsf{pk}), \mathsf{pk}) = 1$. The scheme should also be existentially unforgeable against chosen-message attacks (EUF-CMA) and have perfect adaption (in the vein of [11]).

C Security Model and Proofs

We consider a single revocation, issuing and auditability authority. Extension to the multi-issuing and multi-auditing authorities is straightforward as the corresponding keys can be generated independently. For multiple revocation authorities, one needs to take into account the existence of multiple revocation accumulators and thus adapt the scheme accordingly. As for the issuer-hiding and auditability properties, these are considered independently as extensions.

We denote by Tx the universe of transactions tx represented as bitstrings. Before presenting the oracles and formal definitions, we introduce the following auxiliary lists, sets and global variables. N represents the set of all pseudonyms nym while the sets NYM and RNYM represent the subsets of unrevoked and revoked pseudonyms respectively. Therefore, we have that $\mathsf{NYM} \cap \mathsf{RNYM} = \emptyset \wedge \mathsf{NYM} \cup \mathsf{RNYM} = \mathsf{N}$. $\mathtt{NYM}$ is a list that keeps track of which nym is assigned to which user. The global variables RI and $\mathsf{NYM}_{\mathsf{LoR}}$ (initially set to $\perp$) store the revocation information $(\mathbb{R}_S, \mathbb{R}_V)$ and the pseudonyms used in $\mathcal{O}_{\mathsf{LoR}}$ respectively. The oracles are defined as follows:

$\mathcal{O}_{\mathtt{HU}}(i)$ takes as input a user identity i. If $i \in \mathtt{HU} \cup \mathtt{CU}$, it returns $\perp$. Otherwise, it creates a new honest user i by running $(\mathtt{USK}[i], \mathtt{UPK}[i]) \xleftarrow{\$} \mathsf{UKGen}(\mathsf{opk})$, adding i to the honest user list $\mathtt{HU}$ and returning $\mathtt{UPK}[i]$.

$\mathcal{O}_{\mathtt{CU}}(i, \mathsf{upk})$ takes as input a user identity i and (optionally) upk; if user i does not exist, a new corrupt user with public key upk is registered, while if i is honest, its secret key and all credentials are leaked. If $i \in \mathtt{CU}$, $i \in I_{\mathsf{LoR}}$ (that is, i is a challenge user in the anonymity game) or if $\mathsf{NYM}_{\mathsf{LoR}} \cap \mathsf{N}[i] \neq \emptyset$ then the oracle returns $\perp$. If $i \in \mathtt{HU}$ then the oracle removes i from $\mathtt{HU}$ and adds it to $\mathtt{CU}$; it returns $\mathtt{USK}[i]$ and $\mathtt{CRED}[j]$ for all j with $\mathtt{OWNR}[j] = i$. Otherwise (*i.e.,* $i \notin \mathtt{HU} \cup \mathtt{CU}$), it adds i to $\mathtt{CU}$ and sets $\mathtt{UPK}[i] \leftarrow \mathsf{upk}$.

$\mathcal{O}_{\mathtt{RN}}(\mathsf{rsk}, \mathsf{rpk}, \mathsf{REV})$ takes as input the revocation secret key rsk, the revocation public key rpk and a list REV of pseudonyms to be revoked. If $\mathsf{REV} \cap \mathsf{RNYM} \neq \emptyset$ or $\mathsf{REV} \not\subseteq \mathsf{N}$ return $\perp$. Otherwise, set $\mathsf{RNYM} \leftarrow \mathsf{RNYM} \cup \mathsf{REV}$ and $\mathsf{RI} \leftarrow \mathsf{Revoke}(\mathsf{pp}, (\mathsf{rsk}, \mathsf{rpk}), \mathsf{RNYM}, \mathsf{RI}, 1)$.

$\mathcal{O}_{\mathsf{ObtIss}}(i, \mathcal{X})$ takes as input a user identity i, a pseudonym nym and a set of attributes $\mathcal{X}$. If $i \notin \mathtt{HU}$ or $\exists\, j : \mathtt{NYM}[j] = \mathsf{nym}$, it returns $\perp$. Otherwise, it issues a credential to i by running $(\mathsf{cred}, \top) \xleftarrow{\$} \mathsf{Obtain}(\mathsf{pp}, \mathtt{USK}[i], \mathsf{opk}, \mathsf{apk}, \mathcal{X}, \mathsf{nym})$, $\mathsf{Issue}(\mathsf{pp}, \mathtt{UPK}[i], \mathsf{osk}, \mathsf{apk}, \mathcal{X}, \mathsf{nym})$. If $\mathsf{cred} = \perp$, it returns $\perp$. Else, it appends $(i, \mathsf{cred}, \mathcal{X}, \mathsf{nym})$ to $(\mathtt{OWNR}, \mathtt{CRED}, \mathtt{ATTR}, \mathtt{NYM})$ and returns $\top$.

$\mathcal{O}_{\mathsf{Obtain}}(i, \mathcal{X})$ lets the adversary $\mathcal{A}$, who impersonates a malicious organization, issue a credential to an honest user. It takes as input a user identity i, a pseudonym nym and a set of attributes $\mathcal{X}$. If $i \notin \mathtt{HU}$, it returns $\perp$. Otherwise, it runs $(\mathsf{cred}, \cdot) \xleftarrow{\$} \mathsf{Obtain}(\mathsf{pp}, \mathtt{USK}[i], \mathsf{opk}, \mathsf{apk}, \mathcal{X}, \mathsf{nym}), \cdot)$, where the Issue part is executed by $\mathcal{A}$. If $\mathsf{cred} = \perp$, it returns $\perp$. Else, it appends $(i, \mathsf{cred}, \mathcal{X}, \mathsf{nym})$ to $(\mathtt{OWNR}, \mathtt{CRED}, \mathtt{ATTR}, \mathtt{NYM})$ and returns $\top$.

$\mathcal{O}_{\mathsf{Issue}}(i, \mathcal{X})$ lets the adversary $\mathcal{A}$, who impersonates a malicious user, obtain a credential from an honest organization. It takes as input a user identity i, a pseudonym nym and a set of attributes $\mathcal{X}$. If $i \notin \mathtt{CU}$, it returns $\perp$. Otherwise, it runs $(\cdot, I) \xleftarrow{\$} (\cdot, \mathsf{Issue}(\mathsf{pp}, \mathtt{UPK}[i], \mathsf{osk}, \mathsf{apk}, \mathcal{X}, \mathsf{nym}))$, where the Obtain part is executed by $\mathcal{A}$. If $\mathrm{I} = \perp$, it returns $\perp$. Else, it appends $(i, \perp, \mathcal{X}, \mathsf{nym})$ to $(\mathtt{OWNR}, \mathtt{CRED}, \mathtt{ATTR}, \mathtt{NYM})$ and returns $\top$.

$\mathcal{O}_{\mathsf{Show}}(j, \mathcal{S}, \mathcal{D})$ lets the adversary $\mathcal{A}$ play a dishonest verifier during a showing by an honest user. It takes as input an index of an issuance j and attributes sets $\mathcal{S}$ and $\mathcal{D}$. Let $i \xleftarrow{\$} \mathtt{OWNR}[j]$. If $i \notin \mathtt{HU}$, it returns $\perp$. Otherwise, it runs $(S, \cdot) \xleftarrow{\$} \mathsf{Show}(\mathsf{pp}, \mathtt{USK}[i], \mathtt{UPK}[i], \mathsf{opk}, \mathtt{ATTR}[j], \mathcal{S}, \mathcal{D}, \mathtt{CRED}[j], \mathsf{RI}, \mathsf{apk}, \mathsf{tx}), \cdot)$

$\mathcal{O}_{\mathsf{LoR}}(j_0, j_1, \mathcal{S}, \mathcal{D})$ is the challenge oracle in the anonymity game where $\mathcal{A}$ runs Verify and must distinguish (multiple) showings of two credentials $\mathtt{CRED}[j_0]$ and $\mathtt{CRED}[j_1]$. The oracle takes two issuance indices j_0 and j_1 and attribute sets $\mathcal{S}$ and $\mathcal{D}$. If $J_{\mathsf{LoR}} \neq \emptyset$ and $J_{\mathsf{LoR}} \neq \{j_0, j_1\}$, it returns $\perp$. Let $i_0 \xleftarrow{\$} \mathtt{OWNR}[j_0]$ and $i_1 \xleftarrow{\$} \mathtt{OWNR}[j_1]$. If $J_{\mathsf{LoR}} \neq \emptyset$ then it sets $J_{\mathsf{LoR}} \xleftarrow{\$} \{j_0, j_1\}$ and $I_{\mathsf{LoR}} \xleftarrow{\$} \{i_0, i_1\}$. If $i_0, i_1 \neq \mathtt{HU} \vee \mathsf{N}[i_0] = \perp \vee \mathsf{N}[i_1] = \perp \vee \mathsf{N}[i_0] \in \mathsf{RNYM} \vee \mathsf{N}[i_1] \in \mathsf{RNYM} \vee \mathcal{S} \not\subseteq \mathtt{ATTR}[j_0] \cap \mathtt{ATTR}[j_1] \vee \mathcal{D} \cap \{\mathtt{ATTR}[j_0] \cup \mathtt{ATTR}[j_1]\} \neq \emptyset$, it returns $\perp$. Else, it adds $\mathsf{N}[i_b]$ to $\mathsf{NYM}_{\mathsf{LoR}}$ and runs $(S, \cdot) \xleftarrow{\$} (\mathsf{Show}(\mathsf{pp}, \mathtt{USK}[j_b], \mathtt{UPK}[j_b], \mathsf{opk}, \mathtt{ATTR}[j_b], \mathcal{S}, \mathcal{D}, \mathtt{CRED}[j_b], \mathsf{RI}, \mathsf{apk}, \mathsf{tx}), \cdot)$ (with b set by the experiment)

Unforgeability. An ABC system is unforgeable, if $\forall\ \lambda, q, q' > 0$ and p.p.t adversaries $\mathcal{A}$ having oracle access to $\mathcal{O} := \{\mathcal{O}_{\mathtt{HU}}, \mathcal{O}_{\mathtt{CU}}, \mathcal{O}_{\mathtt{RN}}, \mathcal{O}_{\mathsf{ObtIss}}, \mathcal{O}_{\mathsf{Issue}}, \mathcal{O}_{\mathsf{Show}}\}$ the following probability is negligible.

$$\Pr\left[\begin{array}{l} \mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda, (1^q, 1^{q'})); (\mathsf{rsk}, \mathsf{rpk}) \xleftarrow{\$} \mathsf{RAKGen}(\mathsf{pp}); (\mathsf{ask}, \mathsf{apk}) \xleftarrow{\$} \mathsf{AAKGen}(\mathsf{pp}); \\ (\mathsf{osk}, \mathsf{opk}) \xleftarrow{\$} \mathsf{OKGen}(\mathsf{pp}); (\mathcal{S}, \mathcal{D}, \mathsf{st}) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\mathsf{pp}, \mathsf{opk}, \mathsf{rpk}, \mathsf{apk}); \\ (\cdot, b^*) \xleftarrow{\$} (\mathcal{A}(\mathsf{st}), \mathsf{Verify}(\mathsf{pp}, \mathcal{S}, \mathcal{D}, \mathsf{opk}, \mathsf{rpk}, \mathsf{apk}, \mathsf{RI}, \mathsf{tx}, \Omega)) : \\ b^* = 1 \wedge \forall\ j : \mathtt{OWNR}[j] \in \mathtt{CU} \implies (\mathsf{N}[j] = \perp\ \vee\ (\mathsf{N}[j] \neq \perp\ \wedge (\mathcal{S} \not\subseteq \mathtt{ATTR}[j] \vee \\ \mathcal{D} \subseteq \mathtt{ATTR}[j] \vee \mathsf{N}[j] \in \mathsf{RNYM})) \end{array}\right]$$

Anonymity. An ABC system is anonymous, if $\forall\ \lambda, q, q' > 0$ and all p.p.t adversaries $\mathcal{A}$ having oracle access to $\mathcal{O} := \{\mathcal{O}_{\mathtt{HU}}, \mathcal{O}_{\mathtt{CU}}, \mathcal{O}_{\mathtt{RN}}, \mathcal{O}_{\mathsf{Obtain}}, \mathcal{O}_{\mathsf{Show}}, \mathcal{O}_{\mathsf{LoR}}\}$ the following probability is negligible.

$$\Pr\left[\begin{array}{l} \mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda, (1^q, 1^{q'})); (\mathsf{ask}, \mathsf{apk}) \xleftarrow{\$} \mathsf{AAKGen}(\mathsf{pp}); \\ b \xleftarrow{\$} \{0,1\}; (\mathsf{opk}, \mathsf{rpk}, \mathsf{st}) \xleftarrow{\$} \mathcal{A}(\mathsf{pp}); b^* \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\mathsf{st}) \end{array} : b^* = b\right] - \frac{1}{2}$$

Auditability. An ABC system is auditable if $\forall\ \lambda, q, q' > 0$, $\forall\ \mathcal{X} : 0 < |\mathcal{X}| \leq q$, $\forall\ \emptyset \neq \mathcal{S} \subset \mathcal{X}$, $\forall\ \emptyset \neq \mathcal{D} \not\subseteq \mathcal{X} : 0 < |\mathcal{D}| \leq q$, $\forall\ \mathsf{NYM}, \mathsf{RNYM} \subseteq \mathsf{N} : 0 < |\mathsf{N}| \leq q' \wedge \mathsf{NYM} \cap \mathsf{RNYM} = \emptyset$, $\forall\ \mathsf{nym} \in \mathsf{NYM}$ it holds that:

$\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda, (1^q, 1^{q'}))$; $(\mathsf{ask}, \mathsf{apk}) \xleftarrow{\$} \mathsf{AAKGen}(\mathsf{pp})$; $(\mathsf{rsk}, \mathsf{rpk}) \xleftarrow{\$} \mathsf{RAKGen}(\mathsf{pp})$; $(\mathbb{R}, \mathsf{aux}_{\mathsf{rev}}) \leftarrow \mathsf{RSetup}(\mathsf{pp}, \mathsf{rpk}, \mathsf{NYM}, \mathsf{RNYM})$; $(\mathsf{osk}, \mathsf{opk}) \xleftarrow{\$} \mathsf{OKGen}(\mathsf{pp})$; $(\mathsf{usk}, \mathsf{upk}) \xleftarrow{\$} \mathsf{UKGen}(\mathsf{pp})$;$(\mathsf{cred}, \top) \xleftarrow{\$} (\mathsf{Obtain}(\mathsf{pp}, \mathsf{usk}, \mathsf{opk}, \mathsf{apk}, \mathcal{X}, \mathsf{nym}), \mathsf{Issue}(\mathsf{pp}, \mathsf{upk}, \mathsf{osk}, \mathsf{apk}, \mathcal{X}, \mathsf{nym}))$; $\Omega \leftarrow \mathsf{Show}(\mathsf{pp}, \mathsf{usk}, \mathsf{upk}, \mathsf{opk}, \mathsf{cred}, \mathcal{S}, \mathcal{D}, \mathbb{R}, \mathsf{apk}, \mathsf{tx})$;
$1 \leftarrow \mathsf{Verify}(\mathsf{pp}, \mathcal{S}, \mathcal{D}, \mathbb{R}_V, \mathsf{rpk}, \mathsf{apk}, \mathsf{tx}, \Omega)$; $\mathsf{upk}_1 = \mathsf{AuditDec}(\mathsf{enc}, \mathsf{ask})$

Issuer-Hiding. An ABC system supports issuer-hiding if for all $\lambda > 0$, all $q > 0$, all $n > 0$, all $t > 0$, all $\mathcal{X}$ with $0 < |\mathcal{X}| \leq t$, all $\emptyset \neq \mathcal{S} \subset \mathcal{X}$ and $\emptyset \neq \mathcal{D} \not\subseteq \mathcal{X}$ with $0 < |\mathcal{D}| \leq t$, and p.p.t adversaries $\mathcal{A}$, the following holds.

$$\Pr\left[\begin{array}{l} \mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda, 1^q); \forall\ i \in [n] : (\mathsf{osk}_i, \mathsf{opk}_i) \xleftarrow{\$} \mathsf{OKGen}(\mathsf{pp}); \\ (\mathsf{usk}, \mathsf{upk}) \xleftarrow{\$} \mathsf{UKGen}(\mathsf{pp}); j \xleftarrow{\$} [n]; \\ (\mathsf{cred}, \top) \xleftarrow{\$} (\mathsf{Obtain}(\mathsf{usk}, \mathsf{opk}_j, \mathcal{X}), \mathsf{Issue}(\mathsf{upk}, \mathsf{osk}_j, \mathcal{X})); \\ j^* \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Show}}}(\mathsf{pp}, \mathcal{S}, \mathcal{D}, (\mathsf{opk}_i)_{i \in [n]}) \end{array} : j^* = j\right] \leq \frac{1}{n}$$

Correctness. An ABC system is correct if $\forall\, \lambda > 0$, $\forall\, q, q' > 0$, $\forall\, \mathcal{X} : 0 < |\mathcal{X}| \leq q$, $\forall\, \emptyset \neq \mathcal{S} \subset \mathcal{X}$, $\forall\, \emptyset \neq \mathcal{D} \nsubseteq \mathcal{X} : 0 < |\mathcal{D}| \leq q$, $\forall\, \mathsf{NYM}, \mathsf{RNYM} \subseteq \mathsf{N} : 0 < |\mathsf{N}| \leq q' \wedge \mathsf{NYM} \cap \mathsf{RNYM} = \emptyset$, $\forall\, \mathsf{nym} \in \mathsf{NYM}$, $\forall\, \mathsf{nym}' \in \mathsf{RNYM}$ it holds that:

$\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda, (1^q, 1^{q'}))$; $(\mathsf{rsk}, \mathsf{rpk}) \xleftarrow{\$} \mathsf{RAKGen}(\mathsf{pp})$; $(\mathsf{ask}, \mathsf{apk}) \xleftarrow{\$} \mathsf{AAKGen}(\mathsf{pp})$; $(\mathbb{R}, \mathsf{aux}_{\mathsf{rev}}) \leftarrow \mathsf{RSetup}(\mathsf{pp}, \mathsf{rpk}, \mathsf{NYM}, \mathsf{RNYM})$; $(\mathsf{osk}, \mathsf{opk}) \xleftarrow{\$} \mathsf{OKGen}(\mathsf{pp})$; $(\mathsf{usk}, \mathsf{upk}) \xleftarrow{\$} \mathsf{UKGen}(\mathsf{pp})$; $(\mathsf{cred}, \top) \xleftarrow{\$} (\mathsf{Obtain}(\mathsf{pp}, \mathsf{usk}, \mathsf{opk}, \mathsf{apk}, \mathcal{X}, \mathsf{nym}), \mathsf{Issue}(\mathsf{pp}, \mathsf{upk}, \mathsf{osk}, \mathsf{apk}, \mathcal{X}, \mathsf{nym}))$; $(\mathbb{R}_S, \mathbb{R}_V) \leftarrow \mathsf{Revoke}(\mathsf{pp}, \mathbb{R}, \mathsf{aux}_{\mathsf{rev}}, \mathsf{nym}', 1)$; $\Omega \leftarrow \mathsf{Show}(\mathsf{pp}, \mathsf{usk}, \mathsf{upk}, \mathsf{opk}, \mathsf{cred}, \mathcal{S}, \mathcal{D}, \mathbb{R}, \mathsf{apk}, \mathsf{tx})$; $1 \leftarrow \mathsf{Verify}(\mathsf{pp}, \mathcal{S}, \mathcal{D}, \mathsf{opk}, \mathbb{R}_V, \mathsf{rpk}, \mathsf{apk}, \mathsf{tx}, \Omega)$

Theorem 4. *Protego is correct.*

Proof. It follows by inspection. □

Theorem 5. *If the underlying SPS-EQ perfectly adapts signatures, then Protego is issuer-hiding.*

Proof Sketch. It follows from [11]. □

D NIZK Argument for Issuer-hiding

We refer the reader to [11] (Section 3.1) for the basic syntax and security properties of malleable non-interactive NIZK proof systems.

In Figure 6 we build a fully adaptive malleable NIZK argument following the construction from [11]. The main idea is that given two proofs π_1 and π_2 for statements $\mathbf{x}_1 = w_1\mathbf{v}_i$ and $\mathbf{x}_2 = w_2\mathbf{v}_i$, one can compute a valid proof π for the statement $\mathbf{x} = (\alpha w_1 + \beta w_2)\mathbf{v}_i$ with fresh α and β. The derivation privacy property of the proof system ensures that π looks like a freshly computed proof.

Theorem 6. *The proof system given in Figure 6 is a fully-adaptive NIZK argument for the language $\mathcal{L}_{\bigvee(\mathbf{v}_i)_{i\in[n]}}$, defined as: $\mathcal{L}_{\bigvee(\mathbf{v}_i)_{i\in[n]}} = \{(\mathbf{v}_i) \in \mathbb{G}_2^\ell | \; \exists\, w \in \mathbb{Z}_p^* : \vee\; (\mathbf{v}_i' = w\mathbf{v}_i)_{i\in[n]}\}$.*

Proof Sketch. It follows from [11]. □

E Benchmarks

	$k=2$		$k=4$		$k=6$		$k=8$		$k=10$	
Scheme	Show	Verify	Show	Verify	Show	Verify	Show	Verify	Show	Verify
[7]	173	135	285	258	430	401	599	611	-	-
Protego	148	232	152	233	157	234	163	234	167	237
Protego Duo	66	67	70	69	76	70	80	72	84	74

Table 1: Protocols' comparison showing the running times in milliseconds.

An auditing proof in Protego takes roughly 1.3 and 2.1 ms for the proof generation and verification, surpassing the values from [7]. In Table 2 we report

$\underline{\mathsf{SH.PGen}(1^\lambda)\text{:}}$
$\mathsf{BG} \xleftarrow{\$} \mathsf{BGGen}(1^\lambda);\ z \xleftarrow{\$} \mathbb{Z}_p$
return $(\mathsf{BG}, [z]_1)$
 	
$\underline{\mathsf{SH.PSim}(\mathsf{crs}, \mathsf{td}, (\mathbf{v}_i)_{i\in[n]}, [\mathbf{x}_1]_2, [\mathbf{x}_2]_2)\text{:}}$
$\delta, z_1, ..., z_{n-1} \xleftarrow{\$} \mathbb{Z}_p^*$
$z_n \leftarrow \delta\mathsf{td} - \sum_{i=1}^{i=n-1} z_j$
for all i $\in [n]$ **do**
$\quad d_i \xleftarrow{\$} \mathbb{Z}_p;\ [\mathbf{a}_i]_2 \leftarrow d_i \cdot \mathbf{v}_i - z_i \cdot \mathbf{x}$
return $(([\mathbf{a}_n]_2, [d_n]_1, [z_n]_1)_{n\in[n]}, \delta P_2)$
 	
$\underline{\mathsf{SH.ZKEval}(\mathsf{crs}, [\mathbf{x}_1]_2, [\mathbf{x}_2]_2, \pi; \alpha, \beta)\text{:}}$
$//\ [\mathbf{x}']_2 = (\alpha w_1 + \beta w_2)[\mathbf{v}_i]_2$
$(([\mathbf{a}_n^j]_2, [d_n^j]_1, [z_n]_1)_{n\in[n]}^{j\in[2]}, Z_2) \leftarrow \pi$
$\delta \xleftarrow{\$} \mathbb{Z}_p^*;\ Z_2' \leftarrow \delta Z_2$
for all $i \in [n]$ **do**
$\quad [z_i']_1 \leftarrow \delta[z_i]_1;$
$\quad [d_i']_2 \leftarrow \delta\alpha[d_i^1]_2 + \delta\beta[d_i^2]_2;$
$\quad [\mathbf{a}_i']_2 \leftarrow \delta\alpha[\mathbf{a}_i^1]_2 + \delta\beta[\mathbf{a}_i^2]_2;$
return $(([\mathbf{a}_n']_2, [d_n']_1, [z_n']_1)_{n\in[n]}, Z_2')$
	$\underline{\mathsf{SH.TPGen}(1^\lambda)\text{:}}$
	$\mathsf{BG} \xleftarrow{\$} \mathsf{BGGen}(1^\lambda);\ z \xleftarrow{\$} \mathbb{Z}_p;\ \mathsf{td} \leftarrow z$
	return $(\mathsf{BG}, [z]_1, \mathsf{td})$
	
	$\underline{\mathsf{SH.Prove}(\mathsf{crs}, ([\mathbf{v}_i]_2)_{i\in[n]}, ([\mathbf{x}_j]_2, w_j)_{j\in[2]})\text{:}}$
	$//\ [\mathbf{x}_1]_2 = w_1[\mathbf{v}_i]_2,\ [\mathbf{x}_2]_2 = w_2[\mathbf{v}_i]_2$
	$\delta, r_1, r_2, z_1, ..., z_{n-1} \xleftarrow{\$} \mathbb{Z}_p^*$
	$[z_n]_1 \leftarrow \delta[z]_1 - \sum_{i=1}^{i=n-1}[z_i]_1$
	$([\mathbf{a}_i^j]_2, [d_i^j]_1) \leftarrow (r_j[\mathbf{v}_i]_2, w_j[z_i]_1 + [r_j]_1)$
	for all $k \neq i \in [n],\ j \in [2]$ **do**
	$\quad d_k^j \xleftarrow{\$} \mathbb{Z}_p;\ [\mathbf{a}_k^j]_2 \leftarrow d_k^j[\mathbf{v}_k]_2 - z_k[\mathbf{x}_j]_2$
	return $(([\mathbf{a}_n^j]_2, [d_n^j]_1, [z_n]_1)_{n\in[n]}^{j\in[2]}, \delta P_2)$
	
	$\underline{\mathsf{SH.Verify}(\mathsf{crs}, ([\mathbf{v}_i]_2)_{i\in[n]}, [\mathbf{x}]_2, \pi)\text{:}}$
	$(([\mathbf{a}_n]_2, [d_n]_1, [z_n]_1)_{n\in[n]}, Z_2) \leftarrow \pi$
	check $e([z]_1, Z_2) = e(\sum_{i=1}^{i=n}[z_i]_1, [1]_2)$
	for all i $\in [n]$ **do**
	$\quad$**check** $e([d_i]_1, [\mathbf{v}_i]_2) = e([z_i]_1, [\mathbf{x}]_2) + e([1]_1, [\mathbf{a}_i]_2)$

Fig. 6: Our fully adaptive malleable NIZK argument

the revocation and signing algorithms, including our issuer-hiding NIZK with $n = 5$. For Protego, we consider a signature for vectors of length seven (the size of a credential). In our case, the revocation witnesses are computed by the authority (in linear time) and then randomized by the users (in constant time). For this reason we consider the generation of a single witness for a revocation lists of 10 and a hundred elements (although in practice one would expect it to be closer to 10). For [7], we consider the total time to generate and verify a signature in a user level $L = 2$ (involving two delegations), with revocation times in $\mathbb{G}_2$.

	Revocation				Signature			Issuer-hiding NIZK		
	$n = 10$		$n = 100$		$\ell = 7$ (for Protego)			$n = 5$		
Scheme	Prove	Verify	Prove	Verify	Sign	Verify	ChgRep	Prove	Verify	ZKEval
[7]	88	149	88	149	57	115	N/A	N/A	N/A	N/A
Protego	14	6	140	6	5	16	15	187	179	107

Table 2: Running time for the different algorithms in milliseconds.