# Fully-Featured Anonymous Credentials with Reputation System

Kai Bemmann Johannes Blömer Jan Bobolz Henrik Bröcher
Denis Diemert Fabian Eidens Lukas Eilers Jan Haltermann
Jakob Juhnke Burhan Otour Laurens Porzenheim
Simon Pukrop Erik Schilling Michael Schlichtig
Marcel Stienemeier

September 5, 2018
Department of Computer Science
Paderborn University, Germany
{jbobolz, feidens}@mail.uni-paderborn.de

**Abstract**

We present CLARC (Cryptographic Library for Anonymous Reputation and Credentials), an anonymous credentials system (ACS) combined with an anonymous reputation system. Using CLARC, users can receive attribute-based credentials from issuers. They can efficiently prove that their credentials satisfy complex (access) policies in a privacy-preserving way. This implements anonymous access control with complex policies. Furthermore, CLARC is the first ACS that is combined with an anonymous reputation system where users can anonymously rate services. A user who gets access to a service via a credential, also anonymously receives a review token to rate the service. If a user creates more than a single rating, this can be detected by anyone, preventing users from spamming ratings to sway public opinion.

To evaluate feasibility of our construction, we present an open-source prototype implementation.

**Keywords:** Anonymous Credential System, Reputation System, Privacy, Authentication, Proofs of Partial Knowledge, Revocation

The authors were partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre On-The-Fly Computing (SFB 901).
The authors were partially supported by the Ministry of Education and Research, grant 16SV7055, project "KogniHome".

# 1 Introduction

Usually, authentication is done via identification, i. e. a user supplies his identity and proves possession of some secret in order to gain access to some service or resource. Because user identities are readily available to service providers, providers can exchange collected data about any particular user among themselves.

**Anonymous credential systems** To mitigate such privacy breaches and to give the user more control over her data, one can employ anonymous credential systems (ACS). In an ACS, a user acts under an arbitrary number of unlinkable pseudonyms rather than under his identity. Any two services know a user under different pseudonyms, making it hard to link user data between the two services. A user may even generate multiple pseudonyms for the same service, allowing her to partition generated user data between several unlinkable pseudonyms. In the most extreme case, a user may choose a new pseudonym for every single transaction with any service, making all user actions unlinkable.

Usually, different users have different access rights to some services. In an ACS, these access rights are described by *attributes*. A service provider can issue a *credential* to a user, which is parameterized with attributes. These attributes can, for example, encode access rights to a service or some user data. The user can then *prove possession of a credential* to the same or to other service providers in a privacy-preserving way. This process is called *showing* a credential. This mechanism essentially allows users to carry (authenticated) data and access rights between their different pseudonyms and enables service providers to enforce access restrictions when confronted with anonymous users. Note that in this scenario, the user is in full control of her data and can actively decide what parts of it to reveal to service providers.

**Attributes and policies** A credential may be, for example, used to encode citizen cards issued by the government. Through this credential, the state certifies attributes such as "citizenship", "student status", and "age". The citizen can store this credential, for example, on her smartphone and use it to prove statements about her certified attributes while staying unlinkable across services. The showing of credentials will be done via wireless communications channels of the smartphone, e. g. NFC. As an example, a public transportation provider may provide ticket discounts to students, young people, and senior citizens. To get the discount in this scenario, the user would need to prove possession of a credential whose attributes satisfy a complex policy as in Figure 1. It is a challenge to do this without disclosing the user's specific attribute values to the transportation provider. Note that disclosing (some of) the user's specific attribute values gives the provider quite specific information about the user, which may be used to de-anonymize her. Ideally, the transportation provider only learns a single bit about the attributes, namely that they satisfy the policy.

**Reputation systems** After using the transportation service, the citizen may want to rate her experience using a *reputation system*. On the one hand, the citizen wants to submit her rating anonymously. On the other hand, rated services want to ensure that only people who used the service can rate it. Furthermore, to prevent rating spam, every (anonymous) user should only be

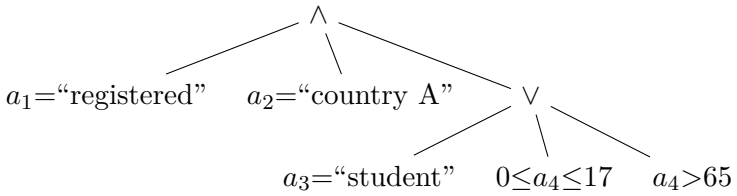


Figure 1: Example (sub-)policy. The attribute $a_1$ stands for the registration status of the user, $a_2$ encodes the citizenship, $a_3$ the student status, and $a_4$ encodes the user's age.

able to submit a single rating. It is a challenge to ensure these properties while simultaneously preserving the user's anonymity.

## 1.1 Contribution

As a solution to the aforementioned challenges, we present our combination of an anonymous credential and reputation system called CLARC (Cryptographic Library for Anonymous Reputation and Credentials). We present the formal construction of the system and the main ideas behind its implementation.

CLARC works as follows: Figure 2 shows the system and involved roles on a high level. We consider one system manager and several users, service providers, and issuers. In the following we describe a usage scenario alongside the numerical order as shown in Figure 2.

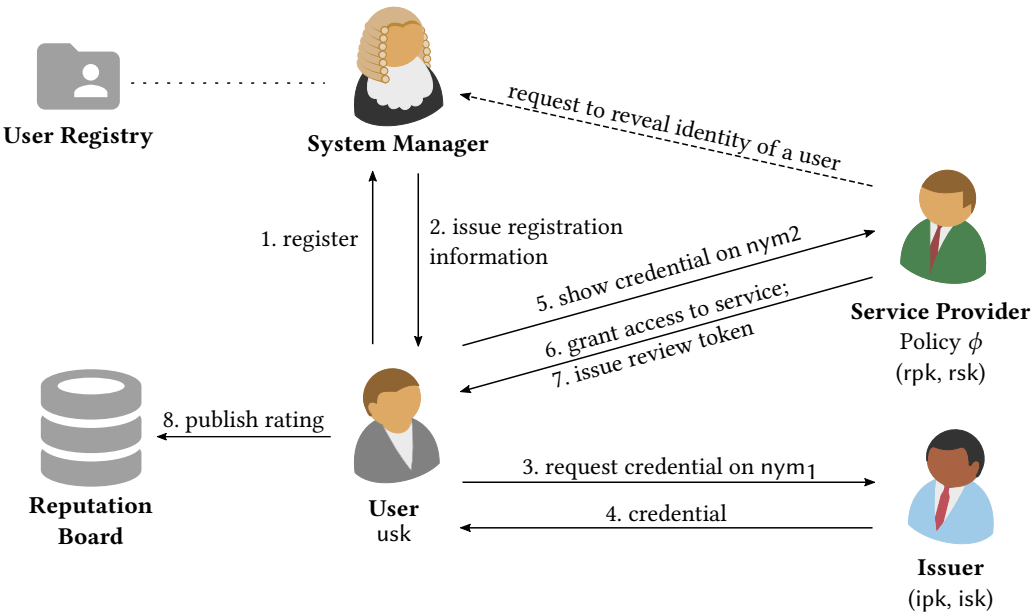


Figure 2: Overview of our system

1.–2. A user has a unique identifier called $\mathsf{usk}$ and registers with the system manager, which is a trusted party. The system manager keeps a registration record of the user and is able to reveal the identity of registered users in case of misuse.

3.–4. A registered user is able to request a credential from an issuer under a pseudonym. The decision whether a request is answered with the issuance of a credential and which attributes are certified depends on the application.

5.–6. Credentials can be shown under a different pseudonym of the same user to service providers. For this, a service provider publishes a policy $\phi$ that determines which combinations of attributes in credentials can access her service.

7.–8. If the service provider accepts the showing of the credential she provides the service and issues a review token to the user. This review token enables the user to rate the provided service once. Reviews are published publicly on a *reputation board*, which is a public storage service. The reviews are anonymous as long as the user is only rates the service once. If a single user rates the same service twice, his two ratings can be publicly linked by anyone. Using this mechanism, the application can detect and decide to ignore duplicate ratings.

The reputation board can be implemented by any public storage system. Ratings cannot be forged or manipulated. However, the storage system should ensure that service providers cannot delete or hide unfavorable ratings. One way to ensure this is by using a public ledger (e. g., a blockchain) as the reputation board.
CLARC supports the following features:

**Access Policies for Service Provider** Service providers can specify complex access policies for their services. A *sub-policy* models the statement "the user has a credential by issuer $\mathsf{ipk}$ with attributes satisfying $\phi$". As such, a sub-policy consists of an issuer's public key and a threshold policy over basic attribute statements. Basic attribute statements in our case are equality ($a_i = k$) and inequality ($a_i \neq k$) of an attribute value $a_i$ to a public value $k$, set membership ($a_i \in S$), and range proofs ($A \leq a_i \leq B$). These basic attribute statements can be arbitrarily connected by threshold gates to form a statement $\phi$ about attributes. In the examples, we sometimes refer to the special case of threshold policies in the form of Boolean formulas (e. g., Figure 1).

A *policy* consists of any number of sub-policies, connected by threshold gates. This allows the service provider to specify policies like "the user has a credential by $\mathsf{ipk}_1$ with attributes satisfying $\phi_1$, or a credential by $\mathsf{ipk}_2$ with attributes satisfying $\phi_2$". In this case, the service provider would not learn whether an authenticated user has been issued sufficient attributes from $\mathsf{ipk}_1$ or from $\mathsf{ipk}_2$.

**Traceability of users** In case of misuse, a trusted third party called *system manager*, can reveal the identity of a credential holder using a special key $\mathsf{osk}$. For more information, see Section 3.4.

**Decentralized validation of ratings** Ratings are published on some (untrusted) *reputation board*. In our system, anyone can verify the validity of ratings.

**Attribute-based ratings** Optionally, users can choose to reveal some credential-authenticated statement about themselves when creating a rating. This enables, for example, the creation of authenticated "expert ratings", where the specific author of the rating remains anonymous (cf. Section 3.6).

**Anonymity** While interacting with users, other entities only learn the information that is revealed by the user (e. g., statements over attributes). Furthermore, a user is anonymous while receiving credentials or rating tokens. Ratings by two buyers of the same product are indistinguishable.

**Credential unforgeability** Users cannot claim to possess credentials that they have not been issued. That is, users cannot claim to satisfy policies for which they have not been issued satisfying credentials.

**Rating authenticity and public linkability** Users cannot rate products unless they have been issued a rating token for it. If a malicious user tries to create two ratings for the same product, the two ratings can be linked publicly.

**Non-impersonation** Attackers cannot successfully claim possession of some honest user's pseudonym (even if they control a service provider or a credential issuer). Furthermore, attackers cannot generate interactions that a system manager would trace to an honest user.

**Open-source prototype implementation** We have implemented CLARC as an open-source Java library under the Apache license. The implementation contains several building blocks that may be of independent interest (cf. Section 5).

## 1.2 Related Work

A standard way to construct an ACS is the framework presented by Camenisch and Lysyanskaya [CL03; Lys02]. Here, a signature scheme, a commitment scheme and efficient protocols suffice to construct a secure ACS. Pairing-based examples for signature schemes that are suitable to be applied in this framework are [CL04; PS16]. Apart from pairing-based systems there also are RSA-based systems, e. g. [CL01; CL03]. There are two techniques to add attributes to credentials. The simplest way is to sign a block of messages, where each message stands for one attribute value. Another way is to map the attribute values to prime numbers. Then, a product of primes is signed to get an attribute-based credential Camenisch and Groß [CG08]. Moreover, Fuchsbauer, Hanser, and Slamanig [FHS14] present attribute-based anonymous credentials supporting selective disclosure using structure preserving signatures on equivalence classes and set commitments.

In comparison to basic credential systems, e. g. [CL03; Lys02], we add attributes to the credentials; a reputation system; expressive policies (cf. Section 4) to the show protocol; revocation and open mechanisms.

Table 1: Comparison of the credential signatures

| | Group | Sig. size | Sec. assmp. |
|---|---|---|---|
| Sec. 3.1 | BN curve [BN06] | $2\,\mathbb{G}_1$ | Assmp. 2 [PS16] |
| Idemix [CV02] | RSA | $1\,\mathbb{Z}_N + 2\,\mathbb{Z}$ | strong RSA [CL03] |
| U-Prove [PZ13] | $\mathbb{G} \subset \mathbb{Z}_p^*$ | $2\,\mathbb{G} + 3\,\mathbb{Z}_q$ | no sec. proof |

Table 2: Comparison of supported relations between credential systems

| | Equality | Inequ. | Set/Range Proof | Threshold Policy | Sel. Disclosure |
|---|---|---|---|---|---|
| Section 3.1 | ✓ | ✓ | ✓ | ✓ | ✓ |
| ABC4Trust [RCS14] | ✓ | ✓ | — | — | ✓ |
| idemix [CV02] | ✓ | ✓ | — | — | ✓ |
| U-Prove [PZ13] | ✓ | ✓ | — | — | ✓ |

The signature scheme proposed by Pointcheval and Sanders [PS16] is a good candidate for the application to anonymous credentials, since the signatures consist of only two group elements, independent of the number of message to be signed. For this reason, we base our construction on [PS16]. Originally, Pointcheval and Sanders proved their signature scheme's security under an interactive assumption. However, Pointcheval and Sanders [PS17] showed recently that the scheme is also secure under a variant of the SDH assumption, which is non-interactive.

There are several implemented credential systems. Among those realizations are Idemix [CV02] and U-Prove [PZ13], both of which are used in the ABC4Trust [RCS14] project. In Table 1, we compare the signatures used as credentials in our system (Section 3.1), idemix and U-Prove. Compared to our CLARC library, the realizations used in ABC4Trust are limited in the sense that they only support disclosing subsets of attribute values when showing a credential (or showing equality of attribute values in multiple credentials). In contrast, CLARC offers more expressive policies (cf. Table 2 for a comparison).

Furthermore, there are two active projects that are focused on bringing credential systems to the end user and cloud providers. These projects are called PRISMACLOUD [and18b] and CREDENTIAL [and18a]. PRISMACLOUD is focused on cryptographic constructions to protect the privacy of users while enabling the usage of cloud services. The CREDENTIAL (Secure Cloud Identity Wallet) project is focused on evaluating and showcasing access control for cloud based services via credential systems. In [Hör+16; Kos+17] the authors focus on the main functions and adoption of the CREDENTIAL concept. Even though the CREDENTIAL concept is based on different cryptographic schemes than our scheme (redactable signature compared to normal signatures) the main concepts (account and identity management) are similar to the one of CLARC.

In addition to anonymous credential systems, anonymous *reputation systems* are of interest. Reputation systems are widely studied in economics and computer science, see for example [And+08; SK13; Del00; DMS03; JI02; KSG03]. Anonymity and security have been identified as key properties of reputation systems and a general formal security definition for reputation

systems was presented by Blömer, Juhnke, and Kolb [BJK15]. In [BJK15] the authors construct a cryptographic reputation system based on group signatures. Additionally the authors [BJK15] add the feature that multiple ratings by one user for one product can be detected. To the best of our knowledge, ours is the first construction of an anonymous reputation and credential system where users are anonymous when authenticating for a service and when rating it. A public-ledger based credential system is presented in [Yan+17]. The system in [Yan+17] also features a reputation system, but there the service providers rate the users after an authentication process.

In previous provable-secure reputation systems [BJK15], users were not anonymous when buying a product (i.e. when receiving a rating token allowing to rate it). Our reputation system is based on credentials and supports public linkability. The linking technique was originally presented in [BJK15].

# 2 Notation

Let $R$ be an NP-relation. Similar to Camenisch and Stadler [CS97], we denote a $\Sigma$ protocol ((interactive three-way) special honest-verifier zero-knowledge proof of knowledge) by $\text{PoK}\{(w) : (x, w) \in R\}$. The instance $x$ and relation $R$ are public knowledge and a prover wants to prove knowledge of witness $w$ to some verifier, without revealing $w$. Similarly, we denote the non-interactive proof obtained by applying the (strong) Fiat-Shamir transformation [FS87; BPW12] to $\text{PoK}\{(w) : (x, w) \in R\}$ by $\text{FS.NIPoK}\{(w) : (x, w) \in R\}$. In the strong Fiat-Shamir transformation, the challenge is generated by hashing the statement and common input.

A signature of knowledge [CL06] for $R$ on some message $m \in \{0, 1\}^*$ obtained by applying the (strong) Fiat-Shamir transformation to $\text{PoK}\{(w) : (x, w) \in R\}$ is denoted by $\text{FS.Sign}\{(w) : (x, w) \in R\}(m)$. The corresponding verification algorithm is denoted by FS.Verify.

# 3 Our Construction

In the following we show the details of our attribute-based anonymous credential and reputation system. First, we show the algorithms and afterwards we explain how the algorithms are mapped to the roles and parties shown in Figure 2. We also present further extensions of our construction to support opening, revocation, and attribute-based ratings.

## 3.1 Algorithm Description

Our system CLARC is based on the ACS framework presented in [CL03; Lys02]. The framework shows how to build ACS with anonymity, unforgeability and non-impersonation from three building-blocks. First a hiding and binding commitment scheme for the pseudonyms. Second, an existentially unforgeable signature scheme for the credentials. Third, protocols for issuing and showing a credential. In CLARC we use Pedersen commitments [Ped92] for the pseudonyms and the pairing-based signature scheme by Pointcheval and Sanders [PS16] to issue credentials. Within the credential showing protocol, we use Schnorr-like $\Sigma$ protocols combined with proofs of partial knowledge [CDS94]. For more details on our protocols, we refer to Section 4.

Let $\lambda \in \mathbb{N}$ be the security parameter. Let $\mathcal{G}$ be a type 3 bilinear group generator, let $\Pi_{\text{Ped}} = (\text{Ped.Setup}, \text{Ped.Com}, \text{Ped.Open})$ be the Pedersen's commitment [Ped92] and let $\Pi^k_{\text{PS}} = (\text{PS.Setup}_k, \text{PS.KeyGen}_k, \text{PS.Sign}_k, \text{PS.Verify}_k)$ be the Pointcheval-Sanders signature scheme for $k$ messages [PS16]. Further, let $H_{\mathbb{G}} \colon \{0,1\}^* \to \mathbb{G}$ be a hash function hashing into a group $\mathbb{G}$. Our system $\Pi$ consists of the following ppt algorithms and protocols.

- $\text{Setup}(1^\lambda)$: Generate a type 3 bilinear group $\mathsf{grp} \coloneqq (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\lambda)$ and pseudonym public parameters $\text{pp}_{\text{nym}} \coloneqq (g_{\text{nym}}, h_{\text{nym}}, p, \mathbb{G}_1)$ with $g_{\text{nym}} \leftarrow \mathbb{G}_1 \setminus \{1\}$ and $h_{\text{nym}} \leftarrow \mathbb{G}_1$. Additionally, create auxiliary public parameters $\text{pp}_{\text{aux}}$ used implicitly in the system, e. g. needed to instantiate certain predicates. Return $\text{pp} \coloneqq (\mathsf{grp}, \text{pp}_{\text{nym}})$.

- $\text{MKeyGen}(\text{pp})$ with $\text{pp} = (\mathsf{grp}, \text{pp}_{\text{nym}}) = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \text{pp}_{\text{nym}})$: Generate key pair $(\mathsf{opk}, \mathsf{osk}) \leftarrow \text{PS.KeyGen}_1(\mathsf{grp})$ for opening, choose public linkability basis $b \leftarrow \mathbb{G}_2 \setminus \{1\}$ and set user registry $\mathsf{reg} \coloneqq \epsilon$. Return $(\mathsf{mpk}, \mathsf{msk}) \coloneqq ((\mathsf{opk}, b), (\mathsf{osk}, \mathsf{reg}))$.

- $\text{UsrInit}(\text{pp}, \mathsf{opk})$ with $\text{pp} = (\mathsf{grp}, \text{pp}_{\text{nym}})$ and opening public key $\mathsf{opk} = (g_{\text{opk}}, Y_{\text{opk}}, \tilde{g}_{\text{opk}}, \tilde{X}_{\text{opk}}, \tilde{Y}_{\text{opk}})$: Choose user secret key $\mathsf{usk} \leftarrow \mathbb{Z}_p$, set user public key $\mathsf{upk} \coloneqq g_{\text{opk}}^{\mathsf{usk}}$ and return $(\mathsf{usk}, \mathsf{upk})$.

- $\text{IssInit}(\text{pp}, 1^\ell)$ with $\text{pp} = (\mathsf{grp}, \text{pp}_{\text{nym}})$ and number of supported attributes $\ell \in \mathbb{N}$: Return issuer key pair $(\mathsf{ipk}, \mathsf{isk}) \leftarrow \text{PS.KeyGen}_{\ell+1}(\mathsf{grp})$.

- $\text{RvwInit}(\text{pp})$ with $\text{pp} = (\mathsf{grp}, \text{pp}_{\text{nym}})$: Return a review token issuer key pair $(\mathsf{rpk}, \mathsf{rsk}) \leftarrow \text{IssInit}(\text{pp}, 1^1)$.

- $\text{CreateNym}(\text{pp}, \mathsf{usk})$ with $\text{pp} = (\mathsf{grp}, \text{pp}_{\text{nym}})$ and user secret $\mathsf{usk}$: Return $(\mathsf{nym}, \mathsf{psk}) \leftarrow \text{Ped.Com}(\text{pp}_{\text{nym}}, \mathsf{usk})$, where $\mathsf{psk} = (\mathsf{usk}, d)$.

- $(\text{Join}(\text{pp}, \mathsf{opk}, \mathsf{usk}, \mathsf{usk}), \text{MJoin}(\text{pp}, \mathsf{opk}, \mathsf{upk}, \mathsf{osk}, \mathsf{reg}))$ is an interactive protocol with common input $\text{pp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, \text{pp}_{\text{nym}})$, opener public key $\mathsf{opk} = (g_{\text{opk}}, Y_{\text{opk}}, \tilde{g}_{\text{opk}}, \tilde{X}_{\text{opk}}, \tilde{Y}_{\text{opk}})$ and $\text{pp}_{\text{nym}} = (g_{\text{nym}}, h_{\text{nym}}, p, \mathbb{G}_1)$. The system master parses the opener secret key $\mathsf{osk}$ as $(x, y)$. The user computes $\tilde{\tau} \coloneqq \tilde{Y}_{\text{opk}}^{\mathsf{usk}}$, sends $\tilde{\tau}$ to the master and runs an argument of knowledge of the form $\text{PoK}\{(\mathsf{usk}) : \mathsf{upk} = g_{\text{opk}}^{\mathsf{usk}}\}$. The master proceeds, if and only if she accepts in this proof *and* $e(\mathsf{upk}, \tilde{Y}_{\text{opk}}) = e(g_{\text{opk}}, \tilde{\tau})$ holds. If there is an entry $(\mathsf{upk}, \sigma, \cdot)$ in $\mathsf{reg}$, the master returns $\sigma$ and stops. Otherwise, the master chooses $u \leftarrow \mathbb{Z}_p^*$, computes

$$\sigma = (\sigma_1, \sigma_2) \coloneqq (g_{\text{opk}}^u, (g_{\text{opk}}^x \cdot \mathsf{upk}^y)^u) \ ,$$

  sends $\sigma$ to the user, and adds the entry $(\mathsf{upk}, \sigma, \tilde{\tau})$ to $\mathsf{reg}$. Then, the user checks the signature using $\text{PS.Verify}_1(\text{pp}, \mathsf{opk}, \mathsf{usk}, \sigma)$. If this check fails, she outputs the error symbol $\perp$, otherwise registration information $\mathsf{reginfo} = (\sigma, \mathsf{opk})$.

- $(\text{ProveNym}(\text{pp}, \mathsf{nym}, \mathsf{psk}), \text{VerifyNym}(\text{pp}, \mathsf{nym}))$ is an interactive protocol with common input $\text{pp} = (\mathsf{grp}, \text{pp}_{\text{nym}})$ with $\text{pp}_{\text{nym}} = (g_{\text{nym}}, h_{\text{nym}}, p, \mathbb{G}_1)$ and pseudonym $\mathsf{nym}$. Then the prover parses the pseudonym secret $\mathsf{psk}$ as $(\mathsf{usk}, d)$ and runs with the verifier $\text{PoK}\{(\mathsf{usk}, d) : \mathsf{nym} = g_{\text{nym}}^d \, h_{\text{nym}}^{\mathsf{usk}}\}$.

- $(\text{RcvCred}(\text{pp}, \mathsf{ipk}, \mathsf{nym}, (a_i)_{i=1}^{\ell}, \mathsf{usk}, \mathsf{psk}), \text{IssCred}(\text{pp}, \mathsf{nym}, (a_i)_{i=1}^{\ell}, \mathsf{isk}))$ is an interactive protocol with common input $\text{pp} = (\mathsf{grp}, \text{pp}_{\text{nym}})$ with $\text{pp}_{\text{nym}} = (g_{\text{nym}}, h_{\text{nym}}, p, \mathbb{G}_1)$, issuer public key $\mathsf{ipk} = (g, Y_0, \ldots, Y_\ell, \tilde{g}, \tilde{X}, \tilde{Y}_0, \ldots, \tilde{Y}_\ell)$ and pseudonym $\mathsf{nym}$. The receiver parses the pseudonym secret $\mathsf{psk}$ as $(\mathsf{usk}, d)$ and generates $pp_{\text{rcv}} = (g, Y_0, p, \mathbb{G}_1)$ and computes $(C, (\mathsf{usk}, r)) \leftarrow \text{Ped.Com}(\text{pp}_{\text{rcv}}, \mathsf{usk})$. Then, she sends $C$ to the issuer and runs

$$\text{PoK}\left\{(\mathsf{usk}, d, r) : \mathsf{nym} = g_{\text{nym}}^{d}\, h_{\text{nym}}^{\mathsf{usk}} \wedge C = g^{r}\, Y_0^{\mathsf{usk}}\right\}$$

  with the issuer. If the proof accepts, the issuer chooses $u \leftarrow \mathbb{Z}_p^*$, computes

$$(\sigma_1', \sigma_2') \coloneqq \left(g^u, \left(g^x\, C \cdot \prod_{i=1}^{\ell} Y_i^{a_i}\right)^u\right)$$

  and sends $(\sigma_1', \sigma_2')$ to the receiver. The receiver computes $\sigma = (\sigma_1', \sigma_2' \cdot (\sigma_1')^{-r})$, and checks the signature's validity via $\text{PS.Verify}_{\ell+1}(\text{pp}, \mathsf{ipk}, (\mathsf{usk}, a_1, \ldots, a_\ell), \sigma)$. If the check fails, she outputs $\perp$, else a credential $\mathsf{cred} = (\sigma, (a_1, \ldots, a_\ell), \mathsf{ipk})$.

- $(\text{RcvToken}(\text{pp}, \mathsf{rpk}, \mathsf{nym}, \mathsf{item}, \mathsf{usk}, \mathsf{psk}), \text{IssToken}(\text{pp}, \mathsf{nym}, \mathsf{item}, \mathsf{rsk}))$ is an interactive protocol, where RcvToken performs the same steps as RcvCred except that it uses $\mathsf{rpk}$ instead of $\mathsf{ipk}$ and $H_{\mathbb{Z}_p}(\mathsf{item})$ instead of $(a_i)_{i=1}^{\ell}$. Analogously, IssToken performs the same steps as IssCred except that it uses $\mathsf{rsk}$ instead of $\mathsf{isk}$, and again, $H_{\mathbb{Z}_p}(\mathsf{item})$ instead of $(a_i)_{i=1}^{\ell}$. After having computed $\sigma$, the user checks the signature's validity via $\text{PS.Verify}_2(\mathsf{rpk}, (\mathsf{usk}, H_{\mathbb{Z}_p}(\mathsf{item})), \sigma)$. If the check fails, she outputs the error symbol $\perp$ and else a review token $\mathsf{token} = (\sigma, \mathsf{item}, \mathsf{rpk})$.

- $(\mathsf{ProveCred}(\text{pp}, \mathsf{opk}, \mathsf{nym}, (\mathsf{ipk}_i, \phi_i)_{i=1}^{m}, \psi, \mathsf{usk}, \mathsf{psk}, \mathsf{reginfo}, \mathsf{cred})_{i=1}^{m}), \mathsf{VerifyCred}(\text{pp}, \mathsf{opk}, \mathsf{nym}, (\mathsf{ipk}, \phi)_{i=1}^{m}, \psi))$ is an interactive protocol with common input $\text{pp} = (\mathsf{grp}, \text{pp}_{\text{nym}})$, $\text{pp}_{\text{nym}} = (g_{\text{nym}}, h_{\text{nym}}, p, \mathbb{G}_1)$, opener public key $\mathsf{opk} = (g_{\text{opk}}, Y_{\text{opk}}, \tilde{g}_{\text{opk}}, \tilde{X}_{\text{opk}}, \tilde{Y}_{\text{opk}})$, pseudonym $\mathsf{nym}$, issuer public keys $\mathsf{ipk}_i = (g^{(i)}, Y_0^{(i)}, \ldots, Y_{\ell_i}^{(i)}, \tilde{g}^{(i)}, \tilde{X}^{(i)}, \tilde{Y}_0^{(i)}, \ldots, \tilde{Y}_{\ell_i}^{(i)})$, and predicates $\phi_i$ and $\psi$. We call a pair $(\mathsf{ipk}_i, \phi_i)$ a *subpolicy*, which consists of an issuer's public key $\mathsf{ipk}_i$ and a predicate $\phi_i$ over attributes. $\psi$ is a monotone threshold formula over subpolicies. The prover parses her private input as $\mathsf{psk} = (\mathsf{usk}, d)$, $\mathsf{reginfo} = ((\hat{\sigma}_1, \hat{\sigma}_2), \mathsf{opk})$, $\mathsf{cred}_i = ((\sigma_1^{(i)}, \sigma_2^{(i)}), (a_1^{(i)}, \ldots, a_{\ell_i}^{(i)}), \mathsf{ipk})$.

  Otherwise, she chooses $s \leftarrow \mathbb{Z}_p^*$, sets $\hat{\sigma}' \coloneqq (\hat{\sigma}_1^s, \hat{\sigma}_2^s)$ and sends $\hat{\sigma}'$ to the verifier. The user computes a Fiat-Shamir heuristic non-interactive argument $\pi$ as seen in Figure 3. This argument can be efficiently instantiated using generalized Schnorr protocols (cf. Section 4).

  The verifier accepts if and only if $\hat{\sigma}_1' \neq 1$ holds and $\pi$ is valid.

- $\text{Open}(\mathsf{opk}, \mathsf{osk}, \mathsf{reg}, t)$ with opening public key $\mathsf{opk} = (g_{\text{opk}}, Y_{\text{opk}}, \tilde{g}_{\text{opk}}, \tilde{X}_{\text{opk}}, \tilde{Y}_{\text{opk}})$, open secret key $\mathsf{osk}$, registry information $\mathsf{reg}$ and a transcript $t = ((\hat{\sigma}_1', \hat{\sigma}_2'), \cdot)$ of a $(\mathsf{ProveCred}, \mathsf{VerifyCred})$ run: Use $\mathsf{VerifyCred}$ to check, whether the transcript is valid. If it is invalid, it outputs the error symbol $\perp$. Otherweise, iterate over the entries $(\mathsf{upk}, \cdot, \tilde{\tau})$ of $\mathsf{reg}$, until $e(\hat{\sigma}_2', \tilde{g}_{\text{opk}})\, e(\hat{\sigma}_1', \tilde{X})^{-1} = e(\hat{\sigma}_1', \tilde{\tau})$ holds. If it finds such an entry, it outputs $\mathsf{upk}$, else $\perp$.

$$\text{FS.NIPoK}\left\{\left(\mathsf{usk}, d, \left((a_j^{(i)}, d_j^{(i)})_{j=1}^{\ell}, \sigma_i, \varphi_i\right)_{i=1}^{m}\right) : \begin{array}{l} \mathsf{nym} = g_{\text{nym}}^{d}\, h_{\text{nym}}^{\mathsf{usk}} \\ \wedge e(\hat{\sigma}_1', \tilde{Y}_{\text{opk}})^{\mathsf{usk}} = \frac{e(\hat{\sigma}_2', \tilde{g}_{\text{opk}})}{e(\hat{\sigma}_1', \tilde{X}_{\text{opk}})} \\ \wedge \varphi_i = 1 \Leftrightarrow (\phi_i(a_1^{(i)}, \ldots, a_{\ell_i}^{(i)}) = 1 \\ \quad \wedge \text{PS.Verify}_{\ell_i+1}(\text{pp}, \mathsf{ipk}_i, \\ \quad\quad (\mathsf{usk}, a_1^{(i)}, \ldots, a_{\ell_i}^{(i)}), \sigma_i) = 1) \\ \wedge \Psi(\varphi_1, \ldots, \varphi_m) = 1 \end{array}\right\}$$

Figure 3: Fiat-Shamir proof used in ProveCred

$$\text{FS.Sign}\left\{(\mathsf{usk}, \zeta, r) : \begin{array}{l} \frac{e(\hat{\sigma}_2', \tilde{g}_{\text{opk}})}{e(\hat{\sigma}_1', \tilde{X}_{\text{opk}})} = e(\hat{\sigma}_1', \tilde{Y}_{\text{opk}})^{\mathsf{usk}} \wedge L_1 = H_{\mathbb{G}_1}(\mathsf{rpk}, \mathsf{item})^{\zeta + \mathsf{usk}} \wedge L_2 = b^{\zeta} \\ \wedge \dfrac{e(\sigma_2', \tilde{g})}{e(\sigma_1', \tilde{X})\, e(\sigma_1', \tilde{Y}_1)^{H_{\mathbb{Z}_p}(\mathsf{item})}} = e(\sigma_1', \tilde{g})^{r}\, e(\sigma_1', \tilde{Y}_0)^{\mathsf{usk}} \end{array}\right\}(m)$$

Figure 4: Signature of knowledge for Rate

- Rate(pp, mpk, rpk, item, reginfo, token, usk, $m$) with pp $=$ (grp, $\text{pp}_{\text{nym}}$), $\text{pp}_{\text{nym}} = (g_{\text{nym}}, h_{\text{nym}}, p, \mathbb{G}_1)$, master public key mpk $=$ (opk, $b$) with opener public key opk $= (g_{\text{opk}}, Y_{\text{opk}}, \tilde{g}_{\text{opk}}, \tilde{X}_{\text{opk}}, \tilde{Y}_{\text{opk}})$, review token issuer public key rpk $= (g, Y_0, Y_1, \tilde{g}, \tilde{X}, \tilde{Y}_0, \tilde{Y}_1)$, item identifier item $\in \{0,1\}^*$, registration information reginfo and review token token, user secret usk and rating text $m \in \{0,1\}^*$: Parse reginfo $= ((\hat{\sigma}_1, \hat{\sigma}_2), \mathsf{opk})$ and token $= ((\sigma_1, \sigma_2), \mathsf{item}, \mathsf{rpk})$. If $\text{PS.Verify}_1(\mathsf{opk}, \mathsf{usk}, (\hat{\sigma}_1, \hat{\sigma}_2)) = 0$ and $\text{PS.Verify}_2(\mathsf{rpk}, (\mathsf{usk}, H_{\mathbb{Z}_p}(\mathsf{item})), (\sigma_1, \sigma_2)) = 0$, output $\perp$ and stop. Otherwise, choose $s \leftarrow \mathbb{Z}_p^*$ and $(u, r) \leftarrow \mathbb{Z}_p^* \times \mathbb{Z}_p$, and set $\hat{\sigma}' := (\hat{\sigma}_1^s, \hat{\sigma}_2^s)$ and $\sigma' := (\sigma_1', \sigma_2') := (\sigma_1^u, (\sigma_2 \cdot \sigma_1^r)^u)$. For public linkability, choose $\zeta \leftarrow \mathbb{Z}_p$ and compute values $L_1 := H_{\mathbb{G}_1}(\mathsf{rpk}, \mathsf{item})^{\zeta + \mathsf{usk}}$ and $L_2 := b^{\zeta}$. Compute $\rho$ as given in Figure 4 and output rating $= ((m, \mathsf{item}), \mathsf{mpk}, \mathsf{rpk}, \hat{\sigma}', \sigma', \rho, L_1, L_2)$. Note that $w := (\mathsf{usk}, \zeta, r)$ is a witness for instance $x := (\text{pp}, \mathsf{mpk}, \mathsf{rpk}, \mathsf{item}, \hat{\sigma}', \sigma')$ of the relation given in Figure 4.

- Verify(pp, mpk, rpk, item, rating, $m$) with pp $=$ (grp, $\text{pp}_{\text{nym}}$), $\text{pp}_{\text{nym}} = (g_{\text{nym}}, h_{\text{nym}}, p, \mathbb{G}_1)$, master public key mpk $=$ (opk, $b$) with the opener's opk $= (g_{\text{opk}}, Y_{\text{opk}}, \tilde{g}_{\text{opk}}, \tilde{X}_{\text{opk}}, \tilde{Y}_{\text{opk}})$, review token issuer public key rpk $= (g, Y_0, Y_1, \tilde{g}, \tilde{X}, \tilde{Y}_0, \tilde{Y}_1)$, item identifier item $\in \{0,1\}^*$, rating $= ((m, \mathsf{item}), (\mathsf{mpk}, \mathsf{rpk}, \hat{\sigma}', \sigma', \rho, L_1, L_2))$ for item and $m \in \{0,1\}^*$: Let FS.Verify be instantiated with the same relation as given in Figure 4. Run FS.Verify$(m, \rho)$ and output the result.

- Link(pp, mpk, rpk, rating, rating$^*$) with public parameters pp $=$ (grp, $\text{pp}_{\text{nym}}$), master public key mpk $=$ (opk, $b$), review token issuer public key rpk, and two ratings rating $=$ $((m, \mathsf{item}), (\mathsf{mpk}, \mathsf{rpk}, \hat{\sigma}', \sigma', \rho, L_1, L_2))$ and rating$^*$ $= ((m^*, \mathsf{item}), (\mathsf{mpk}, \mathsf{rpk}, \hat{\sigma'}^*, \sigma'^*, \rho^*, L_1^*, L_2^*))$ for item: First, check whether rating and rating$^*$ are valid ratings using the Verify. If one of these check fails, output $\perp$ and stop. Otherwise, return 1 if and only if the following equation holds:

$$e\left(\frac{L_1}{L_1^*}, b\right) = e\left(H_{\mathbb{G}_1}(\mathsf{rpk}, \mathsf{item}), \frac{L_2}{L_2^*}\right) \quad .$$

### 3.2 How to use the system

Let $\Pi$ be the credential and reputation system described above. The algorithms of $\Pi$ have to be executed by different parties in the system. One party can act under different roles, e. g. a party can be a service provider and an issuer. In the following we present a usual mapping from algorithms to roles and highlight the interactions between the parties. We refer to Figure 2 for the system overview.

**System Manager** Uses Setup and MKeyGen to generate the public parameters pp and parameters $\mathsf{opk}$ used to support opening. The system manager also takes part in the join/register protocol (Join, MJoin) and executes the MJoin algorithm. In case of misuse a service user can ask the system manger to execute Open and determine which user is responsible.

**User** First, a user has to initialize its parameters and user secret with UsrInit. Next, she has to join at the system manager using Join. A user can derive new pseudonyms from her user secret whenever she wants by using CreateNym. To receive a credential she executes RcvCred of the protocol (RcvCred, IssCred) with the corresponding issuer. She can show that her credentials satisfy some access policy by executing the left-hand side of the ($\mathsf{ProveCred}$, $\mathsf{VerifyCred}$) protocol. The user can also get a review token from a service provider through the execution of (RcvToken, IssToken). The result of RcvToken is a review token bound to the identifier $\mathsf{item}$ of the accessed service. The review token enables the user to form an anonymous review by executing Rate. The message $m$ involved in Rate is the actual review. For example, this can be a textual description or a star rating, depending on the application.

**Issuer** An issuer joins the system by locally executing IssInit. If a user requests a credential the issuer executes the right-hand side of the protocol (RcvCred, IssCred) and can determine which attributes are certified in the credential.

**Service Provider** A normal service provider just needs to publish access policies for her services. If she also wants to issue review tokens to users she executes RvwInit and publishes the corresponding public key $\mathsf{rpk}$. To issue a review token she runs IssToken in the protocol (RcvToken, IssToken) with a user.

**Any Party** Anyone with access to the public parameters of the system can verify published reviews by executing the Verify algorithm. To check whether two ratings for the same service have been created by the same user, anyone can execute the Link algorithm.

### 3.3 Security

The reputation and credential system $\Pi$ as described in Section 3.1 fulfills the following security properties:

**Theorem 3.1.** *If the Pointcheval-Sanders assumption [PS16] and the q-SDH assumption [Ngu05; BB04] hold for* $\mathcal{G}$*, then* $\Pi$ *offers* non-impersonation,credential unforgeability*, and* rating authenticity *(cf. Section 1.1) in the random oracle model.*

**Theorem 3.2.** *If the decisional Diffie-Hellman assumption holds in* $\mathbb{G}_1$*, the Pointcheval-Sanders assumption and the q-SDH assumption hold for* $\mathcal{G}$*, and* $\Pi$ *offers credential unforgeability, then* $\Pi$ *offers* anonymity *(cf. Section 1.1) in the random oracle model.*

Because of space restrictions we omit the formal proofs. The proofs follow the framework presented in [CL03; Cam+15] (for the general construction of credential systems), Pointcheval and Sanders [PS16] (for signature protocols and Open), and Blömer, Juhnke, and Kolb [BJK15] (for public linkability).

### 3.4 Open Mechanism

From the user's point of view anonymity is a desirable property and the main motivation to use an anonymous credential and reputation system. However, anonymity can also be seen as a disadvantage from a service provider's point of view as it protects entities that misbehave. To counteract this issue, we provide the feature of *traceability* in our system. Traceability is one of the classical features of a group signature scheme as defined by Bellare, Micciancio, and Warinschi [BMW03]. Here, it allows the group manager to reveal the identity of the signer of a message. In a credential system we are not interested in the entity that issued the credential in the case of misuse, but the entity that owns the credential, which for example, was used to get the access that resulted in a misbehavior.

In our construction (Section 3.1), the system master can use the algorithm Open to reveal the user public key $\mathsf{upk}$ of some misbehaving user based on a transcript of $(\mathsf{ProveCred}, \mathsf{VerifyCred})$. The particular construction of Open is adapted from the group signature scheme presented by Pointcheval and Sanders in [PS15, Appendix A.1]. To implement this approach, we ask every user willing to use our system to register at the system manager. This is done by running the protocol $(\text{Join}, \text{MJoin})$. By doing so, the system manager builds up a user directory $\mathsf{reg}$ keeping track of every registered user. Note that the system manager during the execution of $(\text{Join}, \text{MJoin})$ only learns $g_{\text{opk}}^{\mathsf{usk}}$ and not $\mathsf{usk}$ itself. The user then additionally needs to show in $(\mathsf{ProveCred}, \mathsf{VerifyCred})$ that she is registered at the system manager. A service provider needs to store every transcript of $(\mathsf{ProveCred}, \mathsf{VerifyCred})$ runs performed—provided the service provider wants to be able to trace a user later on. If the service provider notices a misbehavior (e. g. spamming), she can issue a request at the system manager to open the transcript that was used by the misbehaving user to gain access to her service. In practice, it is required to give a sound reason for this request. Then, the system manager will open the transcript and can reveal the $\mathsf{upk}$ of the misbehaving user.

### 3.5 Revocation

Closely related to the traceability of users described in Section 3.4 is the feature of *revocation*. In the context of our system, revocation means invalidating certain elements. In this section, we present ideas for *revocation of users* and *revocation of credentials*.

### 3.5.1 Revocation of Users

In Section 3.4, we have already seen how we can reveal the identity of a user in the case of misuse. Still, this procedure only reveals the identity without preventing the user from continuing to misbehave. To do so, an option is to exclude the user from the system by revoking the user. A simple, yet inefficient approach, is to let the system manager publish a list of all revoked users, called *revocation list*. In detail, to revoke a user the system manager adds the pair $(\mathsf{upk}, \tilde{\tau})$ generated in the joining phase to the revocation list. Consequently, verifiers need to check whether the user requesting access to a service is not revoked. Therefore, the verifier running VerifyCred in the protocol (ProveCred, VerifyCred) performs the same steps as described in Open but replaces reg by the public revocation list. If the output is not the error symbol $\perp$, the user is classified *non-revoked*. However, as mentioned before, this approach is quite inefficient if many users are revoked, i. e. the size of the revocation list is large.

Note that the check of a user being revoked can be done without interaction with the system manager. This approach therefore is *verifier-local*.

### 3.5.2 Revocation of Credentials

Sometimes it also can be useful for an issuer to revoke certain credentials, e. g. if attributes do not hold anymore for some users. This can be solved similar to the approach presented in Section 3.5.1: An issuer includes a *unique identifier* in the credential. Then, the issuer publishes a list of all credential identifiers that are revoked, called *credential revocation list*. In this scenario, users also need to show in (ProveCred, VerifyCred) that the credential they show is not on the issuer's credential revocation list. To implement this, one can make use of an AND-composition of inequality proofs showing that the identifier contained in the credential is unequal to every element of the credential revocation list. The dynamic accumulator based approach is formalized by Baldimtsi et al. [Bal+17] as an anonymous revocation component that can be added to any anonymous system. Another approach would be to implement a non-membership proof as proposed in [LLX07].

## 3.6 Attribute-Based Ratings

Another extension for our system are *attribute-based ratings*. Although, we aim for anonymity of raters, it is useful to add further information (e. g. about her qualifications) to the rating than only a rating text and the fact that the rater bought the service. For illustration, recall the public transport example given in Section 1: A rating of a seasonal ticket owner is more valuable than a rating of a single-ticket owner, since the former likely uses the public transport more often.

To integrate this feature in our system (Section 3.1) a simple adaption of algorithm Rate suffices. In detail, a user creating a rating integrates a non-interactive proof over some policy (similar to the proof sent in ProveCred) into the rating. The policy represents the information the rater wants to disclose about himself, e.g. "seasonal ticket owner OR 10 years public transport customer".

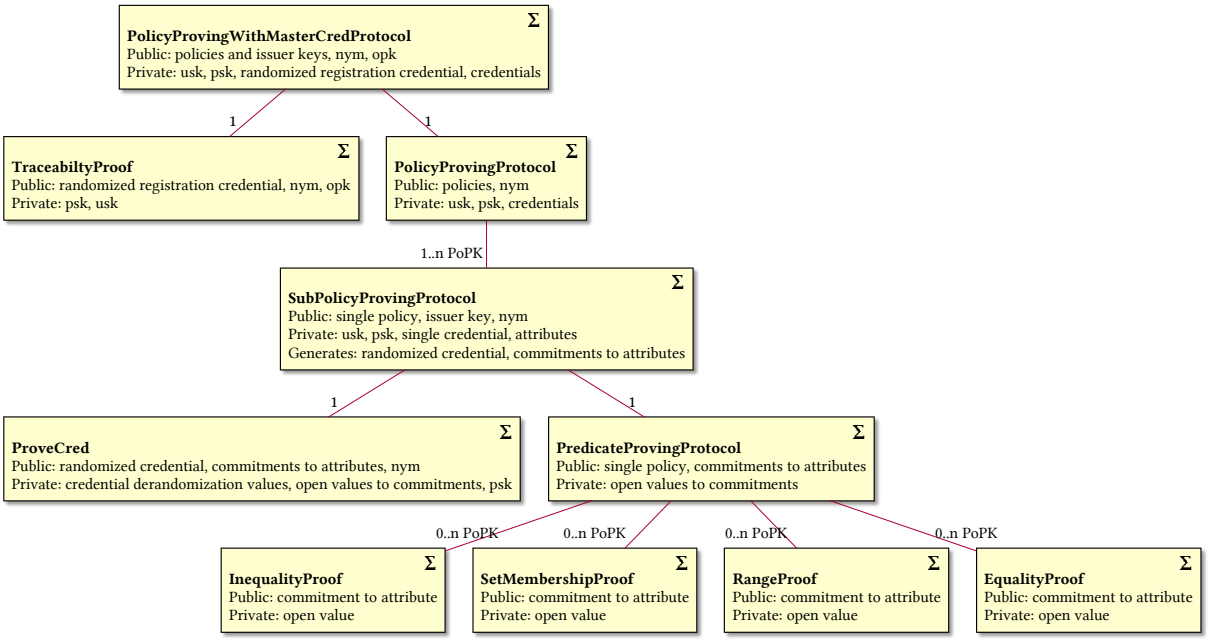


Figure 5: The hierarchical structure of ProveCred. Not pictured: the *inequality*, *set membership*, *equality*, and *range proofs* themselves consist of internal protocols

# 4 Protocol design

When showing a credential, our implementation supports a large class of access policies. For this reason, given some access policy, we need to be able to generate a corresponding protocol proving that the access policy is satisfied. Our approach to generate such protocols is based on the idea of a hierarchical suite of ($\Sigma$) protocols that can be composed ad-hoc as required. The key insight is that almost all relevant protocols can be interpreted as $\Sigma$ protocols with a slightly weakened notion of *soundness* (namely *computational soundness*, see below). This allows us to generically apply standard techniques for $\Sigma$ protocols such as proofs of partial knowledge [CDS94], Damgård's technique [Dam00], and the Fiat-Shamir heuristic [FS87]. An overview of the involved protocols and their (hierarchical) relations is pictured in Figure 5. In this section, we describe this approach by conceptually building a credential proof protocol bottom-up, starting with the protocols on the bottom level of Figure 5.

**Generalized Schnorr protocol** The original Schnorr protocol is a well-known $\Sigma$ protocol allowing to prove knowledge of the discrete logarithm $x$ of a given group element $g^x$. A folklore generalization allows proving knowledge of values $(x_1, \ldots, x_n) \in \mathbb{Z}_p^n$ satisfying

$$\bigwedge_{j=1}^{m} \left( A_j = \prod_{i=1}^{n} g_{i,j}^{x_i} \right)$$

for any given public groups $\mathbb{G}_1, \ldots, \mathbb{G}_m$ of prime order $p$ and public $A_j, g_{i,j} \in \mathbb{G}_j$. We have implemented this construction s.t. given a set of symbolic equations as above, the corresponding generalized Schorr protocol can be derived automatically.

**Basic commitment proofs** We implement protocols that, given a public Pedersen commitment $C$, show that the prover knows values $d, a \in \mathbb{Z}_p$ such that $C = g^a h^d$ and

- $a = a^*$ for some public value $a^*$ (`EqualityProof`)
- $a \neq a^*$ for some public value $a^*$ (`InequalityProof`) [CS03]
- $a \in S^*$ for some public set $S^* \subset \mathbb{Z}_p$ (`SetMembershipProof`) [Ngu05]
- $A^* \leq a_{\text{int}} \leq B^*$ for public values $A^*, B^* \in \mathbb{Z}$ and hidden $a_{\text{int}} \in \mathbb{Z}$ with $0 \leq a_{\text{int}} < p$ and $a_{\text{int}} \in [a]_p$ (`RangeProof`) [Sch01].

The `EqualityProof` can be implemented directly as a generalized Schnorr protocol, proving knowledge of $d$ s.t. $C \cdot g^{-a^*} = h^d$. The other protocols require some more setup. As an example, for the `InequalityProof`, the protocol reads as follows: The prover computes $W = (g^{a-a^*})^z$ with a random $z \leftarrow \mathbb{Z}_p^*$ and sends $W$ to the verifier. She then runs a generalized Schnorr protocol proving knowledge of $a, z, x_{za}, x_{zd}$ such that (1) $1 = g^{x_{za}} h^{x_{zd}} (C^{-1})^z$ and (2) $W = (g^{-a^*})^z g^{x_{za}}$. As a witness for the Schnorr protocol, she uses $a, z$ and $x_{za} = z \cdot a, x_{zd} = z \cdot d$. The verifier accepts iff (3) $W \neq 1$ and the Schnorr protocol accepts. It is easy to see that (1) implies $C = g^{x_{za}/z} h^{x_{zd}/z} \wedge a^* \neq x_{za}/z$ if $z \neq 0$. For $z = 0$ we have $x_{za} \neq 0$ because of (2) and (3), and $0 = \log_h(g) \cdot x_{za} + x_{zd}$ from (1), hence in that case $\log_h(g)$ can be computed from a Schnorr witness. We implemented this two-step protocol (compute W, then Schnorr) as a (single) $\Sigma$ protocol.

For this, we introduce a mechanism we call *protocol wrapping*. Protocol wrapping allows us to wrap two-step protocols into a single (computationally sound) $\Sigma$ protocol. Applied to the inequality proof above, the $\Sigma$ protocol $\Sigma_{\text{ineq}}$ resulting from protocol wrapping works as follows: The prover starts by computing $W$ as above, then she internally instantiates a Schnorr protocol $\Sigma_{\text{Schnorr}}$ for the required equations. Note that the equations depend on the concrete $W$, which is considered public input for the Schnorr protocol. As the first message of $\Sigma_{\text{ineq}}$, the prover sends $W$ and the message $A$, generated by the internal $\Sigma_{\text{Schnorr}}$ as the first message. The verifier chooses her challenge $c$ uniformly at random from $\mathbb{Z}_p$ as usual. The prover's final message for $\Sigma_{\text{ineq}}$ is the final message $R$ of her internal $\Sigma_{\text{Schnorr}}$, given challenge $C$. Upon receiving the final message, the $\Sigma_{\text{ineq}}$ verifier locally instantiates the $\Sigma_{\text{Schnorr}}$ protocol using $W$ received from the prover in the first message. She accepts if $\Sigma_{\text{Schnorr}}$ accepts the transcript $(A, C, R)$ and $W \neq 1$. Note that $\Sigma_{\text{ineq}}$ is a proper (computationally sound) $\Sigma$ protocol for the relation $R_{\text{ineq}} = \{((C, a^*), (a, d)) \mid C = g^a h^d \wedge a \neq a^*\}$: An extractor, given two transcripts $((W, A), C, R)$ and $((W, A), C', R')$ internally runs the Schnorr extractor on $(A, C, R), (A, C', R')$ to retrieve $a, z, x_{za}, x_{zd}$ and then use the observations above to compute a proper witness $(a, d)$ with $C = g^a h^d$ and $a \neq a^*$ or to compute $\log_h(g)$ if $z = 0$. This example motivates our notion of computational soundness: For computationally bounded adversaries, it is hard to come up with two transcripts for which the extractor would *not* output a proper witness for $R_{\text{ineq}}$ (in

this example, finding such transcripts would imply breaking the discrete logarithm assumption). Special honest zero knowledgeness still unconditionally holds for $\Sigma_{\text{ineq}}$.

We apply the protocol wrapping mechanism also for `SetMembershipProof`s (sending a blinded accumulator witness alongside the first message) and `RangeProof`s (sending a blinded accumulator witness for each digit in the $u$-ary representation of $a_{\text{int}}$). As a result of protocol wrapping, all the basic commitment proof protocols introduced in this section are (computationally sound) $\Sigma$ protocols.

**Proving knowledge of a credential satisfying an access policy** Using the protocols above, we can now set up a protocol `SubPolicyProvingProtocol` proving possession of a credential. Specifically, a $\Sigma$ protocol for the relation $\{((\text{pp}, \mathsf{ipk}, \mathsf{nym}, \phi), (\mathsf{usk}, \mathsf{psk}, \mathsf{cred}, \vec{a})) \mid \mathsf{cred}$ is a credential on $\mathsf{usk}$ and attributes $\vec{a}$, and $\phi(\vec{a})\}$. Here, $\phi$ is a monotone predicate over $|\phi|$ atomic expressions of the form "$a_i = a^*$", "$a_i \neq a^*$", "$a_i \in S^*$", or "$A^* \leq a \leq B^*$" (cf. basic commitment proofs). This `SubPolicyProvingProtocol` on the prover side randomizes the credential's signature $\sigma$, yielding $\sigma'$. and generates a Pedersen commitment $C_i = g^{a_i} h^{r_i}$ on each attribute $a_i$. It then sets up a $\Sigma$ protocol `ProveCred` that shows that the prover can derandomize $\sigma'$ to a valid signature on her user secret $\mathsf{usk}$ and the attributes $\vec{a}$ such that the $C_i$ are commitments to the $a_i$ and $\mathsf{nym}$ is a commitment to $\mathsf{usk}$. It furthermore sets up basic commitment proofs $\Sigma_1, \ldots, \Sigma_{|\phi|}$ as in the first paragraph, one for each atomic expression in $\phi$. $\Sigma_j$ for the $j$-th expression proves that some $C_i$ can be opened to an attribute $a_i$ satisfying the $j$-th expression. Finally, the $\Sigma_1, \ldots, \Sigma_{|\phi|}$ are composed using (generalized) proofs of partial knowledge (Theorem 4.1) with access structure according to $\phi$, yielding a (computationally sound) $\Sigma$ protocol `PredicateProvingProtocol`. Let $\Sigma_{\text{composed}}$ be the $\wedge$-composition of `ProveCred` and `PredicateProvingProtocol`. The overall `SubPolicyProvingProtocol` then sends $\sigma', (C_i)_{i=1}^{\ell}$ alongside the first message of $\Sigma_{\text{composed}}$ and runs $\Sigma_{\text{composed}}$ using protocol wrapping.

This construction is enabled by the following theorem, generalized from [CDS94; AAS16].

**Theorem 4.1** (Proofs of partial knowledge)**.** *Let $\Sigma_1, \ldots, \Sigma_n$ be $\Sigma$ protocols for the relations $R_1, \ldots, R_n$, respectively. Let $\Phi$ be a monotone formula $\Phi : \{0,1\}^n \to \{0,1\}$ represented as a circuit consisting of threshold gates. Then, there exists a $\Sigma$ protocol $\Sigma_\Phi$ for the relation $R_\Phi = \{((x_1, \ldots, x_n), (w_1, \ldots, w_n, Z_1, \ldots, Z_n)) \mid (Z_i = 1 \Leftrightarrow (x_i, w_i) \in R_i) \wedge \Phi(Z_1, \ldots, Z_n) = 1\}$. Furthermore, the protocol $\Sigma_\Phi$ can be efficiently computed given $\Sigma_1, \ldots, \Sigma_n$ and $\Phi$.*

In contrast to prior work we allow the composition of several different $\Sigma$ protocols and relations.

**Proving an access policy over multiple credentials** Since `SubPolicyProvingProtocol` is a (computationally sound) $\Sigma$ protocol, multiple such proofs can be connected by arbitrary monotone threshold formulas using proofs of partial knowledge, yielding a protocol `PolicyProvingProtocol`. This allows us, for example, to prove that a user has a credential by issuer $A$ satisfying policy $\phi_A$, or a credential by issuer $B$ satisfying $\phi_B$ without disclosing which of the two the user holds.

**Proving an access policy that requires accountability** A verifier may insist that the anonymous user can be identified by a trusted party in case of dispute. To enable this, we

simply $\wedge$-compose the `PolicyProvingProtocol` with a `TraceabilityProof`. The latter is a $\Sigma$ protocol that proves knowledge of a (randomized) registration credential $\sigma_{\text{reg}}$. The registration credential $\sigma_{\text{reg}}$ is a perfectly binding commitment to the user's `usk` and can be linked to the user's identity by the trusted party (using the special key `osk`). Sending $\sigma_{\text{reg}}$ alongside the first message using protocol wrapping, the `TraceabilityProof` runs a Schnorr proof proving that $\sigma_{\text{reg}}$ is well-formed (essentially guaranteeing that it can be linked by the trusted party) and consistent with the pseudonym `nym`.

**Finalizing the protocol** As the final step, we apply Damgård's technique (for an interactive argument), or the Fiat-Shamir heuristic (for a non-interactive argument or a signature of knowledge, respectively) on the top-level $\Sigma$ protocol `PolicyProvingWithMasterCredProtocol`. The result is a secure argument against computationally bounded verifiers even when many such protocols are run concurrently.

# 5 Implementation

We implemented CLARC in Java 8, building upon the `upb.crypto` library, which offers elliptic curve math and several useful utilities (e.g. hashing). As building blocks for the anonymous credential and reputation system, we implemented Pointcheval-Sanders signatures [PS16], Pedersen's commitment [Ped92] including a hash-then-commit variant, Nguyen's accumulator [Ngu05], basic tree-based Shamir secret sharing [BL90], generalized Schnorr protocols, proofs of partial knowledge [CDS94], Damgård's technique for concurrently black-box secure $\Sigma$ protocols [Dam00], the Fiat-Shamir heuristic [FS87], several general zero-knowledge protocols (see Section 4). These may be of independent interest. For the anonymous credential system, the library offers an easy-to-use lightweight API. This API is conceptually easy to consume and does not require any cryptographic knowledge. However, advanced users can always access lower levels of the library to customize functionality to their use-case. We have evaluated our API and library by implementing a simple example application.

The implementation of CLARC is open-source[1].

## 5.1 Performance

Since the most time critical operations in our system are issuing/receiving a credential and proving that a certain policy is satisfied by a credential, we focus our evaluation on the efficiency of these processes. We considered the following test cases.

1. Equality of a single attribute:
   - *Policy*: `citizenship = "Germany"`
   - *Credential*: certifying only this attribute
2. Range proof over a single attribute:

[1] `https://github.com/upbcuk`

Table 3: Avg. performance of the CLARC implementation on an Intel i7-8650U over 10 runs.

| Policy | (1) | (2) | (3) | (4) | (5) |
|---|---|---|---|---|---|
| IssCred | 25 ms | 25 ms | 40 ms | 31 ms | 27 ms |
| ProveCred | 32 ms | 60 ms | 44 ms | 36 ms | 211 ms |
| VerifyCred | 28 ms | 69 ms | 37 ms | 34 ms | 170 ms |

- *Policy*: `0` $\leq$ `age` $\leq$ `17`
- *Credential*: certifying an age of 16 years

3. AND-composition of two equality proofs over two attributes:
   - *Policy*: `citizenship = "Germany"` $\wedge$ `status = "student"`
   - *Credential*: certifying these two attributes
4. OR-composition of two equality proofs over a single attribute:
   - *Policy*: `status = "student"` $\vee$ `status = "teacher"`
   - *Credential*: certifying student state
5. Complex policy:
   - *Policy*: `citizenship = "Germany"` $\wedge$ `(status = "student"` $\vee$ `0` $\leq$ `age` $\leq$ `17` $\vee$ `age` $\geq$ `66)` $\wedge$ `residence` $\in$ `{"Germany", "Austria", "Switzerland"}`
   - *Credential*: certifying citizenship in Germany, a student state, an age of 20 and residence in Germany

Each of these policies includes a proof of system membership. The timings achieved by our library using the bilinear groups provided by `mcl`[2] (bn256) are presented in Table 3. They do not include time for communication. In our tests, user creation, joining the system, and creating a pseudonym took roughly 200 ms in total.

## References


[AAS16] Hiroaki Anada, Seiko Arita, and Kouichi Sakurai. *Proof of Knowledge on Monotone Predicates and its Application to Attribute-Based Identifications and Signatures*. Cryptology ePrint Archive, Report 2016/483. `http://eprint.iacr.org/2016/483`. 2016.

[and18a] European Union's Horizon 2020 research and innovation programme CREDENTIAL. *CREDENTIAL*. en-US. April 2018. URL: `https://credential.eu/` (visited on April 11, 2018).


[2] mcl: A portable and fast pairing-based cryptography library. `https://github.com/herumi/mcl`

[and18b] European Union's Horizon 2020 research and innovation programme. *PRISMACLOUD – PRIvacy and Security MAintaining services in the CLOUD*. en-US. April 2018. URL: https://prismacloud.eu/ (visited on April 11, 2018).

[And+08] Elli Androulaki, SeungGeol Choi, Steven M. Bellovin, and Tal Malkin. "Reputation Systems for Anonymous Networks". In: *Privacy Enhancing Technologies*. Vol. 5134. LNCS. Springer, 2008, pp. 202–218.

[Bal+17] Foteini Baldimtsi et al. *Accumulators with Applications to Anonymity-Preserving Revocation*. Cryptology ePrint Archive, Report 2017/043. http://eprint.iacr.org/2017/043. 2017.

[BN06] Paulo S. L. M. Barreto and Michael Naehrig. "Pairing-Friendly Elliptic Curves of Prime Order". In: *SAC 2005*. Ed. by Bart Preneel and Stafford Tavares. Vol. 3897. LNCS. Springer, Heidelberg, August 2006, pp. 319–331.

[BMW03] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. "Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions". In: *EUROCRYPT 2003*. Ed. by Eli Biham. Vol. 2656. LNCS. Springer, Heidelberg, May 2003, pp. 614–629.

[BL90] Josh Cohen Benaloh and Jerry Leichter. "Generalized Secret Sharing and Monotone Functions". In: *CRYPTO'88*. Ed. by Shafi Goldwasser. Vol. 403. LNCS. Springer, Heidelberg, August 1990, pp. 27–35.

[BPW12] David Bernhard, Olivier Pereira, and Bogdan Warinschi. "How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios". In: *ASIACRYPT 2012*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. LNCS. Springer, Heidelberg, December 2012, pp. 626–643. DOI: 10.1007/978-3-642-34961-4_38.

[BJK15] Johannes Blömer, Jakob Juhnke, and Christina Kolb. "Anonymous and Publicly Linkable Reputation Systems". In: *FC 2015*. Ed. by Rainer Böhme and Tatsuaki Okamoto. Vol. 8975. LNCS. Springer, Heidelberg, January 2015, pp. 478–488. DOI: 10.1007/978-3-662-47854-7_29.

[BB04] Dan Boneh and Xavier Boyen. "Short Signatures Without Random Oracles". In: *EUROCRYPT 2004*. Ed. by Christian Cachin and Jan Camenisch. Vol. 3027. LNCS. Springer, Heidelberg, May 2004, pp. 56–73.

[Cam+15] Jan Camenisch, Maria Dubovitskaya, Kristiyan Haralambiev, and Markulf Kohlweiss. "Composable and Modular Anonymous Credentials: Definitions and Practical Constructions". In: *ASIACRYPT 2015, Part II*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9453. LNCS. Springer, Heidelberg, November 2015, pp. 262–288. DOI: 10.1007/978-3-662-48800-3_11.

[CG08] Jan Camenisch and Thomas Groß. "Efficient attributes for anonymous credentials". In: *ACM CCS 08*. Ed. by Peng Ning, Paul F. Syverson, and Somesh Jha. ACM Press, October 2008, pp. 345–356.

[CL01] Jan Camenisch and Anna Lysyanskaya. “An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation”. In: *EUROCRYPT 2001*. Ed. by Birgit Pfitzmann. Vol. 2045. LNCS. Springer, Heidelberg, May 2001, pp. 93–118.

[CL03] Jan Camenisch and Anna Lysyanskaya. “A Signature Scheme with Efficient Protocols”. In: *SCN 02*. Ed. by Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano. Vol. 2576. LNCS. Springer, Heidelberg, September 2003, pp. 268–289.

[CL04] Jan Camenisch and Anna Lysyanskaya. “Signature Schemes and Anonymous Credentials from Bilinear Maps”. In: *CRYPTO 2004*. Ed. by Matthew Franklin. Vol. 3152. LNCS. Springer, Heidelberg, August 2004, pp. 56–72.

[CS03] Jan Camenisch and Victor Shoup. “Practical Verifiable Encryption and Decryption of Discrete Logarithms”. In: *CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. LNCS. Springer, Heidelberg, August 2003, pp. 126–144.

[CS97] Jan Camenisch and Markus Stadler. “Efficient Group Signature Schemes for Large Groups (Extended Abstract)”. In: *CRYPTO’97*. Ed. by Burton S. Kaliski Jr. Vol. 1294. LNCS. Springer, Heidelberg, August 1997, pp. 410–424.

[CV02] Jan Camenisch and Els Van Herreweghen. “Design and Implementation of The Idemix Anonymous Credential System”. In: *ACM CCS 02*. Ed. by Vijayalakshmi Atluri. ACM Press, November 2002, pp. 21–30.

[CL06] Melissa Chase and Anna Lysyanskaya. “On Signatures of Knowledge”. In: *CRYPTO 2006*. Ed. by Cynthia Dwork. Vol. 4117. LNCS. Springer, Heidelberg, August 2006, pp. 78–96.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. “Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols”. In: *CRYPTO’94*. Ed. by Yvo Desmedt. Vol. 839. LNCS. Springer, Heidelberg, August 1994, pp. 174–187.

[Dam00] Ivan Damgård. “Efficient concurrent zero-knowledge in the auxiliary string model”. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2000, pp. 418–430.

[Del00] Chrysanthos Dellarocas. “Immunizing Online Reputation Reporting Systems Against Unfair Ratings and Discriminatory Behavior”. In: *EC 2000*. ACM, 2000, pp. 150–157.

[DMS03] Roger Dingledine, Nick Mathewson, and Paul Syverson. “Reputation in P2P Anonymity Systems”. In: *Workshop on Economics of Peer-to-Peer Systems*. Vol. 92. 2003.

[FS87] Amos Fiat and Adi Shamir. “How to Prove Yourself: Practical Solutions to Identification and Signature Problems”. In: *CRYPTO’86*. Ed. by Andrew M. Odlyzko. Vol. 263. LNCS. Springer, Heidelberg, August 1987, pp. 186–194.

[FHS14] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. *Structure-Preserving Signatures on Equivalence Classes and Constant-Size Anonymous Credentials*. Cryptology ePrint Archive, Report 2014/944. `http://eprint.iacr.org/2014/944`. 2014.

[Hör+16] Felix Hörandner, Stephan Krenn, Andrea Migliavacca, Florian Thiemer, and Bernd Zwattendorfer. "CREDENTIAL: A Framework for Privacy-Preserving Cloud-Based Data Sharing". In: *ARES*. IEEE Computer Society, 2016, pp. 742–749.

[JI02] Audun Jøsang and Roslan Ismail. "The Beta Reputation System". In: *BLED 2002*. 2002, pp. 41–55.

[KSG03] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. "The EigenTrust Algorithm for Reputation Management in P2P Networks". In: *WWW 2003*. ACM, 2003, pp. 640–651.

[Kos+17] Alexandros Kostopoulos et al. "Towards the Adoption of Secure Cloud Identity Services". In: *ARES*. ACM, 2017, 90:1–90:7.

[LLX07] Jiangtao Li, Ninghui Li, and Rui Xue. "Universal Accumulators with Efficient Nonmembership Proofs". In: *ACNS 07*. Ed. by Jonathan Katz and Moti Yung. Vol. 4521. LNCS. Springer, Heidelberg, June 2007, pp. 253–269.

[Lys02] Anna A. Lysyanskaya. "Signature Schemes and Applications to Cryptographic Protocol Design". PhD thesis. Cambridge, MA, USA, 2002.

[Ngu05] Lan Nguyen. "Accumulators from Bilinear Pairings and Applications". In: *CT-RSA 2005*. Ed. by Alfred Menezes. Vol. 3376. LNCS. Springer, Heidelberg, February 2005, pp. 275–292.

[PZ13] Christian Paquin and Greg Zaverucha. *U-Prove Cryptographic Specification V1.1 (Revision 3)*. December 2013. URL: `https://www.microsoft.com/en-us/research/publication/u-prove-cryptographic-specification-v1-1-revision-3/`.

[Ped92] Torben P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *CRYPTO'91*. Ed. by Joan Feigenbaum. Vol. 576. LNCS. Springer, Heidelberg, August 1992, pp. 129–140.

[PS15] David Pointcheval and Olivier Sanders. *Short Randomizable Signatures*. Cryptology ePrint Archive, Report 2015/525. `http://eprint.iacr.org/2015/525`. 2015.

[PS16] David Pointcheval and Olivier Sanders. "Short Randomizable Signatures". In: *CT-RSA 2016*. Ed. by Kazue Sako. Vol. 9610. LNCS. Springer, Heidelberg, February 2016, pp. 111–126. DOI: `10.1007/978-3-319-29485-8_7`.

[PS17] David Pointcheval and Olivier Sanders. *Reassessing Security of Randomizable Signatures*. Cryptology ePrint Archive, Report 2017/1197. `https://eprint.iacr.org/2017/1197`. 2017.

[RCS14] Kai Rannenberg, Jan Camenisch, and Ahmad Sabouri. *Attribute-based Credentials for Trust: Identity in the Information Society*. Springer, 2014.

[SK13] Sebastian Claußand Stefan Schiffner and Florian Kerschbaum. "k-anonymous Reputation". In: *ASIA CCS 2013*. ACM, 2013, pp. 359–368.

[Sch01] Berry Schoenmakers. "Some efficient zeroknowledge proof techniques". Slides presented at the *International Workshop on Cryptographic Protocols*. March 2001.

[Yan+17] Rupeng Yang, Man Ho Au, Qiuliang Xu, and Zuoxia Yu. *Decentralized Blacklistable Anonymous Credentials with Reputation*. Cryptology ePrint Archive, Report 2017/389. `http://eprint.iacr.org/2017/389`. 2017.