A Formal Security Analysis of Hyperledger AnonCreds

Ashley Fraser[1] and Steve Schneider[2]

[1] School of Computing and Communications, Lancaster University, UK
a.fraser5@lancaster.ac.uk
[2] Surrey Centre for Cyber Security, University of Surrey, UK
s.schneider@surrey.ac.uk

Abstract. In an anonymous credential system, users collect credentials from issuers, and can use their credentials to generate privacy-preserving identity proofs that can be shown to third-party verifiers. Since the introduction of anonymous credentials by Chaum in 1985, there has been promising advances with respect to system design, security analysis and real-world implementations of anonymous credential systems.
In this paper, we examine Hyperledger AnonCreds, an anonymous credential system that was introduced in 2017 and is currently undergoing specification. Despite being implemented in deployment-ready identity system platforms, there is no formal security analysis of the Hyperledger AnonCreds protocol. We rectify this, presenting syntax and a security model for, and a first security analysis of, the Hyperledger AnonCreds protocol. In particular, we demonstrate that Hyperledger AnonCreds is correct, and satisfies notions of unforgeability and anonymity. We conclude with a discussion on the implications of our findings, highlighting the importance of rigorous specification efforts to support security evaluation of real-world cryptographic protocols.

1 Introduction

Anonymous credential systems [16, 23] enable users to collect credentials from issuers and show their credentials to third-party verifiers. Crucially, credentials are *anonymous*, meaning that different credential showings cannot be linked. In particular, issuers cannot determine when credentials are shown by users to third-parties. Since their introduction, a considerable number of anonymous credential systems have been designed e.g., [5, 19, 38, 48, 43]. One such system is the Hyperledger AnonCreds (HLAC) protocol, which first appeared in 2017 and is currently hosted by the Linux Foundation Decentralized Trust [30], and undergoing specification [26] by the Hyperledger AnonCreds Working Group [47].

The HLAC protocol has been used in the design and implementation of real-world credential systems. Notably, the Hyperledger Aries library [28], a toolkit for building decentralised identity management systems, embeds the HLAC protocol. The Hyperledger Aries library, and subsequently the HLAC protocol, has been used in several deployed identity management systems. In particular, the

Government of British Columbia used Hyperledger Aries to create OrgBook [40], a public directory of registered organisations and related data. Moreover, the International Air Transport Association (IATA) launched a pilot scheme for a travel pass in 2021 that enabled travellers to collect documentation related to their flight and vaccination status in one place, simplifying the airport process.

Despite the specification, and the real-world implementations, of the HLAC protocol, it has not undergone any formal security analysis. Accordingly, in this paper, we provide a first formal description of the protocol in the literature, and introduce syntax and a formal security model for the HLAC protocol. We demonstrate security of the HLAC protocol, with the aim that our work will be a first step towards demonstrating the utility and security of this promising protocol.

1.1 Related Work

We provide an overview of related work, namely anonymous credentials. As the anonymous credentials literature is very wide and varied, we focus on the literature most relevant to the HLAC protocol, and refer the reader to [39] for a detailed and systematic review of anonymous credentials.

Anonymous Credential Systems. Anonymous credentials were first introduced in [23] and, in [16], Camenisch and Lysyanskaya introduced the first practical anonymous credential scheme. This scheme uses a signature scheme that has since been generalised [18] to the RSA-based CL signature scheme [18], Pedersen commitments [42] and a series of non-interactive zero-knowledge (NIZK) proof systems. Though other approaches to anonymous credential system design exist, for example, the use of self-blindable credentials [2, 25, 48] and attribute-based signature [45, 49], the design of [16] is the basis for IBM's Idemix protocol [37, 43] and many other anonymous credential systems, e.g., [19, 20, 38]. Moreover, the Hyperledger AnonCreds protocol is based on the design in [16], with NIZK proofs as described in [43].

Many variations on anonymous credential systems exist, including credentials that enable revocation of anonymity [16], delegatable credentials [4, 9, 21], updatable credentials [10] and issuer-hiding credentials [11, 24]. One other feature that many anonymous credential systems aim to achieve, and which is a key feature of the HLAC protocol is *revocation* of credentials. There are a number of approaches to the provision of revocation mechanisms in anonymous credential systems. The most common approach, and the one used in the HLAC protocol, makes use of a cryptographic accumulator scheme [7, 41], which provides an efficient way to maintain a set, and for users to prove membership of it, without disclosing the individual members of the set, thus preserving anonymity [17]. Other approaches to revocation include the use of Credential-Revocation Lists [12, **?**] and the use of issuer-controlled attributes within anonymous credentials, specifically validity time [15]. The HLAC protocol adopts the cryptographic accumulator in [14] that is designed for revocation of anonymous credentials.

Security of Anonymous Credential Systems. Early security models for anonymous credential systems presented a real-ideal world paradigm that captures a single notion of security [5, 16], but, more recently, the literature has focused on game-based security models, e.g., [22, 25, 36], providing individual security experiments for each security requirement. Security models have typically defined two key properties of anonymous credential systems: anonymity [4, 9, 24, 31, 38, 45] and unforgeability [1, 4, 9, 11, 16, 22, 24, 39]. Further properties have been defined, including all-or-nothing transferability [16], unlinkability [1, 11, 16, 38, 39, 45] and soundness [9].

The game-based approach to security definitions has been used in the analysis of anonymous credential systems, e.g., [11, 32, 44], and facilitates the clear mapping of intuitive security notions to formal definitions of security. As this paper is a first look at the HLAC protocol, we adopt the approach of [11, 32, 44], focusing on foundational properties for anonymous credential systems. Specifically, we present a game-based security model that captures anonymity and unforgeability.

Verifiable Credentials. As specified in a W3C Recommendation [46], verifiable credentials enable holders to collect credentials that can be used to produce identity proofs that can be verified by third-party verifiers. While they share similarities with anonymous credentials, verifiable credentials need not generate privacy-preserving proofs, though they do support the use of anonymity mechanisms such as zero-knowledge proofs. The HLAC protocol, often considered a verifiable credential system, embeds many of the concepts of verifiable credentials as set out in the W3C Recommendation, but does not currently comply with the data model of the W3C Recommendation. Accordingly, in this paper, we refer to the HLAC protocol as an *anonymous* credential system, not a *verifiable* credential system.

1.2 Our Contributions

In this paper, we present the first formalisation and security analysis of the HLAC protocol. Though the protocol and its security goals were described in [8], the description was informal. In contrast, in this paper, we present formal definitions of unforgeability and anonymity, and full security proofs. Namely, in Section 2.1, we introduce syntax for the HLAC protocol and provide full technical details of our security model. Our informal notions of security align with the underlying intuition of existing game-based definitions of security for anonymous credential protocols, e.g., [11, 32, 44], but are adapted to the HLAC syntax. We describe the building blocks used in the protocol in Section 3 and describe the HLAC protocol in Section 4. We then demonstrate that the HLAC protocol satisfies correctness, unforgeability and anonymity, presenting full security proofs in Section 5. We provide concluding remarks in Section 6.

Looking forward, we believe that the results presented in this work are meaningful to the real-world security guarantees of the HLAC protocol. We prove that HLAC satisfies key security properties that any anonymous credential system

should achieve, building confidence in the security of HLAC. Furthermore, evaluation of the cryptographic protocol outlined in the specification ensures that our analysis applies to the design of the protocol rather than a specific implementation. Hence, our analysis is relevant to the security of any future implementation. We also hope that the formalisation presented in this paper can support completion of the specification. In particular, we note that, as specification efforts are ongoing, several details, particularly those related to revocation of credentials, are missing from the specification. To address this, we consulted the reference implementation [26] and academic literature outlining the primitives used in the HLAC protocol.

2 Syntax and Security Model

In this section we present a formal framework for the analysis of the Hyperledger AnonCreds (HLAC) protocol [26]. Specifically, we introduce the syntax for the HLAC protocol, and a formal security model for our syntax. First, we provide a high-level overview of the actors and phases of the protocol.

The HLAC protocol involves three actors as follows:

1. *Issuers*: issue credentials to holders. They generate AnonCreds objects, e.g., credential schemas and related cryptographic data, to support the issuance and revocation of credentials.
2. *Holders*: collect and manage their credentials in a digital wallet. Holders obtain credentials from issuers in an interactive process, and can use issued credentials to generate identity proofs that can be shared with verifiers.
3. *Verifiers*: request identity proofs from holders and verify the validity of identity proofs, including the revocation status of credentials used to generate the identity proof.

To support credential issuance and identity proof verification, the HLAC protocol is supported by a *ledger*, a verifiable data registry that stores AnonCreds objects (i.e., credential schemas, credential definitions and revocation registries) that all actors have read access to. Additionally, issuers can write to the ledger, posting AnonCreds objects. In this way, any actor, including holders and verifiers, can obtain data about credentials that is crucial to support the issuance of credentials and verification of identity proofs.

The HLAC protocol typically proceeds in three phases, as shown in Figure 1. In the **setup** phase, the ledger is initialised as empty, and holders initialise an empty credential list and generate a master secret. Issuer setup usually requires the creation, and posting to the ledger, of three AnonCreds objects. That is, the issuer creates a schema and credential definition for each credential that they wish to issue. To issue revocable credentials, an issuer must additionally create a revocation registry for each credential definition. Briefly, a credential schema defines the credential attributes and the credential definition (resp., revocation registry) defines the cryptographic key pair and related data used to issue (resp., demonstrate the revocation status of) a credential. The protocol then proceeds

to the **credential issuance** phase, an interactive process in which a holder provides the issuer with attribute values, and the issuer generates and issues a credential, and updates the revocation status of the credential. Finally, in the **proof presentation** phase, holders can use their issued credentials to construct identity proofs. In this phase, verifiers can specify the (types of) credentials that holders can use to satisfy the proof request and can, accordingly, designate the issuers that they are willing to accept credentials from. As is typical in an anonymous credential system, holders in the HLAC protocol do not need to reveal their entire credentials to verifiers. Rather, holders can reveal a subset of attribute values or reveal facts (i.e., a predicate) about their attribute values, without revealing the attributes themselves. For example, a holder that has a credential stating their date of birth can generate an identity proof that reveals their date of birth without revealing any other credential attributes, or prove that they are over 18 years of age without revealing their date of birth attribute.

Note that, though we describe the HLAC protocol in three consecutive phases, new actors can join the system at any time, creating, issuing, storing and verifying data as required by their role. In addition, actors may execute the credential issuance and proof presentation phases out of order, for example, when an issuer wishes to convince themselves of the accuracy and validity of a holder's attribute values before issuing a credential.

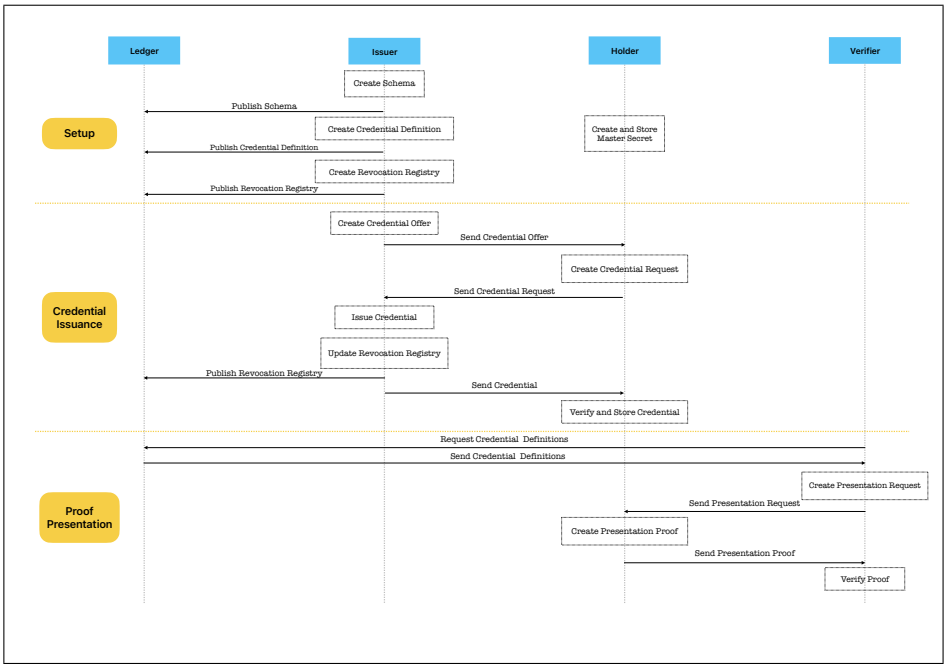

Fig. 1: A typical data flow between a single issuer, holder and verifier in the HLAC protocol.

2.1 Syntax

We now introduce the formal syntax for the HLAC protocol. We first define setup algorithms, and then introduce syntax for the credential issuance and proof presentation phases. We also define algorithms that enable entities to read from, and write to, the ledger $\mathtt{L}$.

Notation. In our syntax, the empty set is written as $\emptyset$ and the cardinality of set S is written as $|S|$. We denote the set of integers as $\mathbb{Z}$ and let $\mathbb{Z}_p$ denote the set $\{1, \ldots, p-1\}$. We write $[n]$ to denote the set $\{1, \ldots, n\}$. We denote an empty list as $()$, write $L \leftarrow L||y$ to denote appending y to the list L, and write $L[i]$ to denote the value of list L at position i. We write $x \leftarrow X$ to denote assignment of X to x. We write $y \leftarrow\!\!\$\, Y$ to denote choosing an element from set Y uniformly at random and assigning it to the variable y. Moreover, we denote $y \leftarrow A(x_1, \ldots, x_n)$ as the result of running deterministic algorithm A on inputs $x_1, \ldots, x_n$. We denote failure of an algorithm as $\perp$. We write $A^{\mathcal{O}}$ to denote algorithm A with access to an oracle $\mathcal{O}$. We say that a function $f\colon \mathbb{N} \to \mathbb{R}$ is negligible if, for every positive polynomial p, there exists an N such that for all integers $n > N$, $f(n) < 1/p(n)$. We denote an arbitrary negligible function as $\mathsf{negl}(\lambda)$ for some security parameter λ.

Setup. We define LSetup that initialises an empty ledger as follows.

- $\mathsf{LSetup}()$: Algorithm LSetup outputs an empty ledger $\mathtt{L} = ()$.

Holders run algorithm HSetup to initialise an empty credential list and generate a master secret.

- $\mathsf{HSetup}()$: Algorithm HSetup generates a master secret $\mathtt{ms}$ and initialises an empty list of credentials $\mathtt{W}$, outputting $(\mathtt{ms}, \mathtt{W})$.

We define algorithms Schema and DSetup that are run by issuers to create a schema and a credential definition respectively. In a typical use case, the issuer generates both the schema and credential definition, with the schema simply being a precursor to the credential definition. However, issuers can reuse an existing schema (i.e., one created by a different issuer) to create a credential definition. However, the issuer must run DSetup for each credential that they want to issue.

- $\mathsf{Schema}(\{\mathtt{a}_2, \ldots, \mathtt{a}_L\})$: On input a set of $L-1$ attributes $\{\mathtt{a}_2, \ldots, \mathtt{a}_L\}$, algorithm Schema generates a schema ID $\mathtt{ID}_\mathtt{S}$ and a credential schema $\mathtt{S}$, and outputs $(\mathtt{ID}_\mathtt{S}, \mathtt{S})$.
- $\mathsf{DSetup}(\mathtt{ID}_\mathtt{S}, \mathtt{S}, \mathtt{b})$: On input a Schema ID $\mathtt{ID}_\mathtt{S}$, schema $\mathtt{S}$, and revocation flag $\mathtt{b}$, algorithm DSetup generates a public/private credential definition pair $(\mathtt{pDef}, \mathtt{sDef})$ and a credential definition ID $\mathtt{ID}_\mathtt{D}$. If $\mathtt{b} = 1$, the credential pair includes material used in the revocation of credentials. Algorithm DSetup outputs $(\mathtt{ID}_\mathtt{D}, \mathtt{pDef}, \mathtt{sDef})$.

Note that, in the HLAC protocol, the first attribute of any credential is reserved. Accordingly, in our syntax, algorithm Schema takes as input $L-1$ attributes that are defined by the issuer. We will provide further details on the reserved attribute in our description of the HLAC protocol (Section 4), but, briefly, the first attribute in any credential is a blinded version of the holder's master secret, tying the holder to the issued credential.

To issue revocable credentials, an issuer must create a revocation registry and revocation list for each credential definition. We define algorithm RSetup for this purpose. We also define algorithm RUpdate that is used to update the revocation list, which typically occurs when credentials are issued or revoked.

- $\mathsf{RSetup}(\mathtt{ID_D}, \mathtt{max})$: On input a credential definition ID $\mathtt{ID_D}$ and a limit on the number of revocable credentials that can be issued $\mathtt{max}$, algorithm RSetup generates a public/private revocation registry pair $(\mathtt{pReg}, \mathtt{sReg})$, a revocation list $\mathtt{A}$, and IDs for the public revocation registry $\mathtt{ID_R}$ and list $\mathtt{ID_A}$, and outputs $(\mathtt{ID_R}, \mathtt{pReg}, \mathtt{sReg}, \mathtt{ID_A}, \mathtt{A})$.
- $\mathsf{RUpdate}(\mathtt{A}, \mathtt{pReg}, \mathtt{sReg}, \mathtt{C_I}, \mathtt{C_R})$: On input a revocation registry list $\mathtt{A}$, public revocation registry $\mathtt{pReg}$, private revocation registry $\mathtt{sReg}$ and sets $\mathtt{C_I} \in \{0,1\}^{\mathtt{max}}$ and $\mathtt{C_R} \in \{0,1\}^{\mathtt{max}}$ that indicate credentials that must be issued and revoked respectively, algorithm RUpdate generates a new revocation list $\mathtt{A'}$ and a new list ID $\mathtt{ID_A}$ and outputs $(\mathtt{ID_A}, \mathtt{A'})$.

Credential Issuance. We define four algorithms to capture the interactive credential issuance process between an issuer and a holder as follows.

- $\mathsf{Offer}(\mathtt{ID_S}, \mathtt{ID_D}, \mathtt{sDef})$: Run by the issuer, on input a schema ID $\mathtt{ID_S}$, credential definition ID $\mathtt{ID_D}$ and a private credential definition $\mathtt{sDef}$, algorithm Offer outputs a credential offer $\mathtt{Offer}$.
- $\mathsf{Request}(\mathtt{Offer}, \mathtt{ms}, \{v_2, \ldots, v_L\})$: Run by the holder, on input a credential offer $\mathtt{Offer}$, a master secret $\mathtt{ms}$ and a set of attribute values $\{v_2, \ldots, v_L\}$, algorithm Request outputs a public/private credential request pair $(\mathtt{pReq}, \mathtt{sReq})$.
- $\mathsf{Issue}(\mathtt{pReq}, \mathtt{Offer}, \mathtt{pDef}, \mathtt{sDef}, \mathtt{pReg}, \mathtt{sReg}, \mathtt{ID_A}, \mathtt{A}, \mathtt{S}, \mathtt{i})$: Run by the issuer, on input a credential request $\mathtt{pReq}$, credential offer $\mathtt{Offer}$, public/private credential definition pair $(\mathtt{pDef}, \mathtt{sDef})$, public/private revocation registry pair $(\mathtt{pReg}, \mathtt{sReg})$, ID of a revocation list $\mathtt{ID_A}$, revocation list $\mathtt{A}$, schema $\mathtt{S}$ and index $\mathtt{i}$, algorithm Issue outputs a credential $\mathtt{cred}$.
- $\mathsf{Store}(\mathtt{cred}, \mathtt{pReq}, \mathtt{sReq}, \mathtt{W})$: Run by the holder, on input a credential $\mathtt{cred}$, a credential request $\mathtt{pReq}$, a private credential request $\mathtt{pReq}$ and a credential list $\mathtt{W}$, algorithm Store outputs the updated credential list such that $\mathtt{W} \leftarrow \mathtt{W} \| \mathtt{cred}$.

Proof Presentation. We capture a verifier that requests and verifies identity proofs, and a holder that generates identity proofs in our syntax as follows.

- $\mathsf{Propose}(\mathtt{R}, \mathtt{P})$: Run by the verifier, on input a set of attributes to be revealed $\mathtt{R}$ and a set of predicates $\mathtt{P}$, algorithm Propose outputs a presentation request $\mathtt{PReq}$.

– $\mathsf{Present}(\mathtt{PReq}, \mathsf{C_P}, \mathtt{ms})$: Run by the holder, on input a presentation request $\mathtt{PReq}$, a list of credentials that will be used in the proof $\mathsf{C_P}$, and a master secret $\mathtt{ms}$, algorithm $\mathsf{Present}$ outputs an identity proof $\mathtt{Proof}$.
– $\mathsf{Verify}(\mathtt{Proof}, \mathtt{PReq})$: On input the identity proof $\mathtt{Proof}$ and the presentation request $\mathtt{PReq}$, algorithm Verify returns 1 if the proof verifies and 0 otherwise.

Ledger Functions. We define functions LPost and $\mathsf{LRetrieve}$ to respectively write to and read from the ledger. In practice, algorithms Schema, DSetup, RSetup and $\mathsf{RUpdate}$ internally run algorithm LPost to generate IDs for AnonCreds objects.

– $\mathsf{LPost}(\mathtt{L}, m)$: On input a verifiable data registry $\mathtt{L}$ and a message m, algorithm LPost generates an ID for message m, ID_m, updates $\mathtt{L}$ with m such that $\mathtt{L} \leftarrow \mathtt{L} \| m$, and returns ID_m.
– $\mathsf{LRetrieve}(\mathsf{ID}_m)$: On input an ID for a message m, algorithm $\mathsf{LRetrieve}$ returns the entry associated with ID_m from the verifiable data registry $\mathtt{L}$.

Correctness. An anonymous credentials system must satisfy *correctness*, which requires that, if credentials are issued by executing the setup and credential issuance phase honestly, algorithm Store will update the holder's credential list with the issued credential. Moreover, it must be possible to use credentials stored in the credential list to construct verifiable identity proofs, if the stored credentials "satisfy" a presentation request $\mathtt{PReq}$. To formalise this second requirement, we introduce function $\psi\colon (\mathtt{PReq}, \mathcal{X}) \to \{0,1\}$ and say that credential list $\mathcal{X}$ can be used to construct an identity proof that will satisfy $\mathtt{PReq}$, if $\psi(\mathtt{PReq}, \mathcal{X}) = 1$. Otherwise, $\mathcal{X}$ does not satisfy $\mathtt{PReq}$. We now provide a formal definition of correctness that captures this intuition.

Definition 1 (Correctness). *Let* $\{\mathsf{a}_2, \ldots, \mathsf{a}_L\}$ *be a set of attributes and* $\{v_2, \ldots, v_L\}$ *be a set of corresponding attribute values. Let* $\mathtt{R}$ *and* $\mathtt{P}$ *be sets, and* $\mathtt{PReq}$ *be a presentation request such that* $\psi(\mathtt{PReq}, \mathtt{W}) = 1$ *for an honestly generated credential list* $\mathtt{W}$ *that encodes* $\{(\mathsf{a}_i, v_i)\}_{i \in [2,L]}$. *Then, a* HLAC *construction satisfies* correctness *if there exists a negligible function* negl *such that*

$$\Pr\left[\begin{array}{c}
\mathtt{L} \leftarrow \mathsf{LSetup}() \\
(\mathtt{ms}, \mathtt{W}) \leftarrow\$ \mathsf{HSetup}() \\
(\mathtt{ID_S}, \mathtt{S}) \leftarrow\$ \mathsf{Schema}(\{\mathtt{a}_2, \ldots, \mathtt{a}_L\}) \\
(\mathtt{ID_D}, \mathtt{pDef}, \mathtt{sDef}) \leftarrow\$ \mathsf{DSetup}(\mathtt{ID_S}, \mathtt{S}, 1) \\
(\mathtt{ID_R}, \mathtt{pReg}, \mathtt{sReg}, \mathtt{ID_A}, \mathtt{A}) \leftarrow\$ \mathsf{RSetup}(\mathtt{ID_D}, \mathtt{max}) \\
\mathtt{Offer} \leftarrow\$ \mathsf{Offer}(\mathtt{ID_S}, \mathtt{ID_D}, \mathtt{sDef}) \\
(\mathtt{pReq}, \mathtt{sReq}) \leftarrow\$ \mathsf{Request}(\mathtt{Offer}, \mathtt{ms}, \{v_2, \ldots, v_L\}) \\
\mathtt{cred} \leftarrow\$ \mathsf{Issue}(\mathtt{pReq}, \mathtt{Offer}, \mathtt{pDef}, \mathtt{sDef}, \mathtt{pReg}, \mathtt{sReg}, \mathtt{ID_A}, \mathtt{A}, \mathtt{S}, \mathtt{i}) \\
(\mathtt{ID_A}, \mathtt{A}') \leftarrow\$ \mathsf{RUpdate}(\mathtt{A}, \mathtt{pReg}, \mathtt{sReg}, \{\mathtt{i}\}, \emptyset) \\
\mathtt{W} \leftarrow \mathsf{Store}(\mathtt{cred}, \mathtt{pReq}, \mathtt{sReq}, \mathtt{W}) \\
\mathtt{PReq} \leftarrow\$ \mathsf{Propose}(\mathtt{R}, \mathtt{P}) \\
\mathtt{Proof} \leftarrow\$ \mathsf{Present}(\mathtt{PReq}, \mathtt{W}, \mathtt{ms}) \\
\textbf{return } (\mathtt{cred} \in \mathtt{W} \wedge \mathsf{Verify}(\mathtt{Proof}, \mathtt{PReq}) = 1)
\end{array}\right] \geq 1 - \mathsf{negl}.$$

2.2 Security Model

We now define a security model for our syntax that captures unforgeability and anonymity. Informally, unforgeability requires that an identity proof must

pertain to credentials issued by the issuers that registered the corresponding credential definitions (i.e., posted the credential definitions to the ledger). In addition, the identity proof must be created by the holder to whom the credentials were issued. Anonymity requires that a verifier should not learn which holder generated an identity proof.

Our security definitions are inspired by existing security notions for anonymous credential schemes that are captured in the well-established game-based model of security, e.g., [10, 31, 43]. In this setting, we consider the following threat model.

- *Issuers*: malicious issuers may attempt to break anonymity of holders, and may collude with other issuers and verifiers to do so.
- *Holders*: malicious holders may try to generate identity proofs not supported by their issued credentials.
- *Verifiers*: verifiers may collude with other actors to break holder anonymity or forge credentials.

We now define oracles for our security experiments, and then describe our unforgeability and anonymity experiments.

Oracles. We define several oracles for our security experiments in Figures 2, 3 and 4. At the beginning of each experiment, an oracle $\mathcal{O}$Setup is run to initialise several sets and list for the experiments, and to initialise the ledger. For example, we define list $\mathbf{H}$ to be the list of holder master secrets and write that $\mathbf{H}[i]$ is the master secret of holder registered at index $\mathtt{i}$. We similarly define lists $\mathbf{L}$, $\mathbf{D}$, $\mathbf{R}$ and $\mathbf{C}$ to respectively store credential lists, credential definition data, revocation indices and credential request data. Additionally, $\mathcal{O}$Setup initialises sets QH, QCH and QP to keep track of registered holders, corrupt holders and queries to the proof oracle respectively.

The adversary can perform setup operations by calling oracles $\mathcal{O}$RegH and $\mathcal{O}$RegCDef. In particular, oracle $\mathcal{O}$RegCDef runs algorithms Schema and DSetup to create a schema and credential definition for a set of attributes $\{\mathtt{a}_2, \ldots, \mathtt{a}_L\}$ input to the oracle. If input flag $b = 1$, the oracle also runs RSetup to create a revocation registry for the credential. The IDs for all algorithm outputs is returned by the oracle. Oracle $\mathcal{O}$RegH performs holder setup operations. That is, on input an index i, $\mathcal{O}$RegH runs algorithm HSetup and returns $\top$ if setup was successful. We also provide an oracle $\mathcal{O}$CorruptH that can be used to corrupt holders, returning the master secret and credential list for holder i. We define oracle $\mathcal{O}$Proof to model the proof presentation phase of the HLAC protocol. In particular, $\mathcal{O}$Proof runs algorithm $\mathsf{Present}$ to generate and return an identity proof $\mathtt{Proof}$ for a presentation request $\mathtt{PReq}$ on behalf of holder i.

We then define three oracles in Figure 3 to model the credential issuance process: $\mathcal{O}$CredIssuance, $\mathcal{O}$Offer and $\mathcal{O}$Issue. Oracle $\mathcal{O}$CredIssuance simulates the full credential issuance process, running algorithms Offer, $\mathsf{Request}$, Issue and Store to issue a credential corresponding to $\mathtt{ID}_\mathtt{D}$ to holder j for a set of attribute values V_j. In this way, the experiment models credential issuance where both the holder and credential issuer are honest. In our unforgeability and anonymity

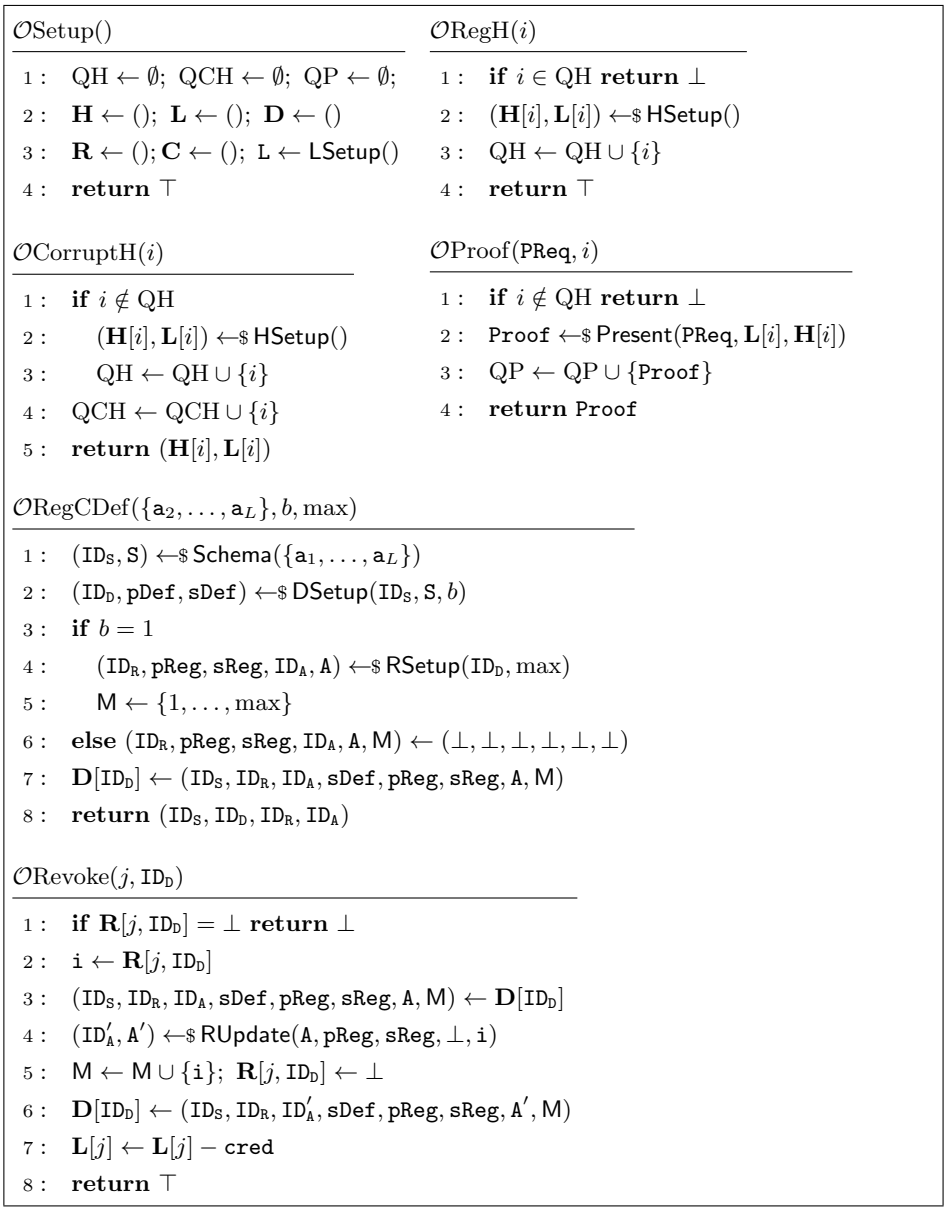

Fig. 2: Oracles used in the unforgeability and anonymity experiments.

experiments, we model an attacker that can corrupt holders, running algorithms Request and Store on behalf of corrupt holders. Accordingly, we define oracles $\mathcal{O}$Offer and $\mathcal{O}$Issue to model an attacker that obtains credential offers and issued credentials from an honest issuer, on behalf of a corrupt holder. As expected, $\mathcal{O}$Offer and $\mathcal{O}$Issue run algorithms Offer and Issue respectively. We also define

$\mathcal{O}\mathrm{Offer}(\mathtt{ID}_\mathtt{D})$

$(\mathtt{ID}_\mathtt{S}, \mathtt{ID}_\mathtt{R}, \mathtt{ID}_\mathtt{A}, \mathtt{sDef}, \mathtt{pReg}, \mathtt{sReg}, \mathtt{A}, \mathsf{M}) \leftarrow \mathbf{D}[\mathtt{ID}_\mathtt{D}]$
$\mathtt{Offer} \leftarrow\!\$\ \mathsf{Offer}(\mathtt{ID}_\mathtt{S}, \mathtt{ID}_\mathtt{D}, \mathtt{sDef})$
return $\mathtt{Offer}$

$\mathcal{O}\mathrm{CredIssuance}(\mathtt{ID}_\mathtt{D}, \mathsf{V}_j, j)$

if $j \notin \mathrm{QH}$ **return** $\bot$
$(\mathtt{ID}_\mathtt{S}, \mathtt{ID}_\mathtt{R}, \mathtt{ID}_\mathtt{A}, \mathtt{sDef}, \mathtt{pReg}, \mathtt{sReg}, \mathtt{A}, \mathsf{M}) \leftarrow \mathbf{D}[\mathtt{ID}_\mathtt{D}]$
$\mathtt{Offer} \leftarrow\!\$\ \mathsf{Offer}(\mathtt{ID}_\mathtt{S}, \mathtt{ID}_\mathtt{D}, \mathtt{sDef})$
$(\mathtt{pReq}, \mathtt{sReq}) \leftarrow\!\$\ \mathsf{Request}(\mathtt{Offer}, \mathbf{H}[j], \mathsf{V}_j)$
if $\mathtt{ID}_\mathtt{R} \neq \bot$
  $\mathtt{i} \leftarrow\!\$\ \mathsf{M}$; $\mathsf{M} \leftarrow \mathsf{M} \setminus \{\mathtt{i}\}$
  $(\mathtt{ID}'_\mathtt{A}, \mathtt{A}') \leftarrow\!\$\ \mathsf{RUpdate}(\mathtt{A}, \mathtt{pReg}, \mathtt{sReg}, \mathtt{i}, \bot)$
  $\mathbf{D}[\mathtt{ID}_\mathtt{D}] \leftarrow (\mathtt{ID}_\mathtt{S}, \mathtt{ID}_\mathtt{R}, \mathtt{ID}'_\mathtt{A}, \mathtt{sDef}, \mathtt{pReg}, \mathtt{sReg}, \mathtt{A}', \mathsf{M})$
  $\mathbf{R}[j, \mathtt{ID}_\mathtt{D}] \leftarrow \mathtt{i}$
else $\mathtt{i} \leftarrow \bot$
$\mathtt{cred} \leftarrow\!\$\ \mathsf{Issue}(\mathtt{pReq}, \mathtt{Offer}, \mathtt{pDef}, \mathtt{sDef}, \mathtt{pReg}, \mathtt{sReg}, \mathtt{ID}_\mathtt{A}, \mathtt{A}, \mathtt{S}, \mathtt{i})$
$\mathbf{L}[j] \leftarrow \mathsf{Store}(\mathtt{cred}, \mathtt{pReq}, \mathtt{sReq}, \mathbf{L}[j])$
return $\top$

$\mathcal{O}\mathrm{Issue}(\mathtt{ID}_\mathtt{D}, \mathtt{pReq}, \mathtt{Offer}, j)$

if $j \notin \mathrm{QCH}$ **return** $\bot$
$(\mathtt{ID}_\mathtt{S}, \mathtt{ID}_\mathtt{R}, \mathtt{ID}_\mathtt{A}, \mathtt{sDef}, \mathtt{pReg}, \mathtt{sReg}, \mathtt{A}, \mathsf{M}) \leftarrow \mathbf{D}[\mathtt{ID}_\mathtt{D}]$
if $\mathtt{ID}_\mathtt{R} \neq \bot$
  $\mathtt{i} \leftarrow\!\$\ \mathsf{M}$; $\mathsf{M} \leftarrow \mathsf{M} \setminus \{\mathtt{i}\}$; $\mathbf{R}[j, \mathtt{ID}_\mathtt{D}] \leftarrow \mathtt{i}$
  $(\mathtt{ID}^*_\mathtt{A}, \mathtt{A}^*) \leftarrow\!\$\ \mathsf{RUpdate}(\mathtt{ID}_\mathtt{A}, \mathtt{sReg}, i, \bot)$
  $\mathbf{D}[\mathtt{ID}_\mathtt{D}] \leftarrow (\mathtt{ID}_\mathtt{S}, \mathtt{ID}_\mathtt{R}, \mathtt{ID}'_\mathtt{A}, \mathtt{sDef}, \mathtt{pReg}, \mathtt{sReg}, \mathtt{A}', \mathsf{M})$
else $\mathtt{i} \leftarrow \bot$
$\mathtt{cred} \leftarrow\!\$\ \mathsf{Issue}(\mathtt{pReq}, \mathtt{Offer}, \mathtt{pDef}, \mathtt{sDef}, \mathtt{pReg}, \mathtt{sReg}, \mathtt{ID}_\mathtt{A}, \mathtt{A}, \mathtt{S}, \mathtt{i})$
$\mathbf{L}[j] \leftarrow \mathbf{L}[j] \| \mathtt{cred}$
return $\mathtt{cred}$

Fig. 3: Additional oracles used in the unforgeability and anonymity experiments.

an oracle $\mathcal{O}$Revoke that runs algorithm $\mathsf{RUpdate}$ to revoke a credential issued to holder j and corresponding to $\mathtt{ID}_\mathtt{D}$.

Our anonymity experiment captures an attacker that can corrupt issuers, in addition to holders. To model this, we define three further oracles, formalised in Figure 4. Firstly, oracle $\mathcal{O}$CorruptCDef models corruption of an issuer, returning a private credential definition $\mathtt{sDef}$ and private revocation registry $\mathtt{sReg}$ for

a credential definition input to the oracle. Oracles $\mathcal{O}$Request and $\mathcal{O}$Store run algorithms Request and Store respectively. Together, these oracles enable an adversary to play the role of a corrupt issuer, obtaining credential requests from, and storing credentials on behalf of, honest holders.

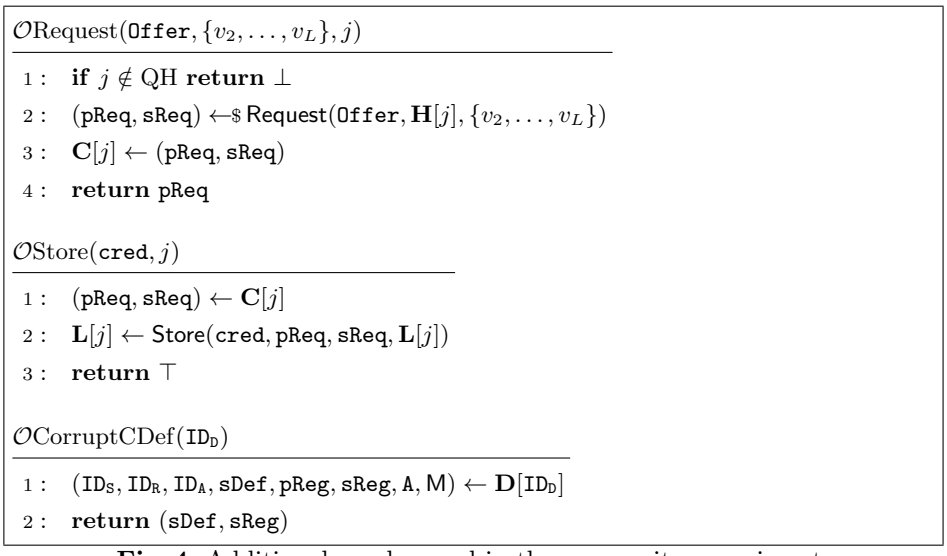

Fig. 4: Additional oracles used in the anonymity experiment.

Unforgeability. Unforgeability requires that a holder can only construct identity proofs for credentials issued to the holder by issuers that posted associated credential definitions to the ledger. In other words, an attacker cannot forge credentials on behalf of an honest issuer that can be used to construct valid identity proofs, even if the attacker can corrupt credential holders. Formally, we define an experiment (Definition 2) in which an adversary attempts to output a presentation request $\texttt{PReq}^*$ and a proof $\texttt{Proof}^*$ such that algorithm Verify returns 1. In the experiment, the adversary can query the oracles defined in Figure 2. We say that the HLAC protocol satisfies unforgeability if the adversary outputs a valid identity proof with negligible probability and the following two conditions hold. Firstly, the identity proof $\texttt{Proof}^*$ output by the adversary cannot be the output of oracle $\mathcal{O}$Proof. This is crucial to ensure that the adversary does not trivially succeed in the experiment. Secondly, for every corrupt holder $j \in$ QCH, the credential list $\mathbf{L}[j]$ cannot satisfy the presentation request. This condition ensures that the adversary does not succeed by returning an identity proof pertaining to credentials held by a single corrupt holder, which the adversary can trivially do as they have the corrupt holder's credential list and master secret.

Definition 2 (Unforgeability). *The* HLAC *protocol satisfies* unforgeability *if, for any probabilistic polynomial time adversary $\mathcal{A}$, there exists a negligible func-*

tion negl *such that*

$$\Pr\left[\begin{array}{c} \top \leftarrow \mathcal{O}\mathrm{Setup}();\ (\mathtt{PReq}^*, \mathtt{Proof}^*) \leftarrow\!\$\ \mathcal{A}^{\mathcal{O}}() \\ \textbf{return}\ \big(\mathsf{Verify}(\mathtt{PReq}^*, \mathtt{Proof}^*){=}1\ \wedge\ \mathtt{Proof}^* \notin \mathrm{QP} \\ \wedge\ \{\psi(\mathtt{PReq}^*, \mathbf{L}[j]){=}0\}_{j \in \mathrm{QCH}}\big) \end{array}\right] \leq \mathsf{negl}$$

where $\mathcal{O} = \{\mathcal{O}\mathrm{RegH}, \mathcal{O}\mathrm{CorruptH}, \mathcal{O}\mathrm{RegCDef}, \mathcal{O}\mathrm{CredIssuance}, \mathcal{O}\mathrm{Offer}, \mathcal{O}\mathrm{Issue}, \mathcal{O}\mathrm{Proof}, \mathcal{O}\mathrm{Revoke}\}$ *are the oracles defined in Figures 2 and 3.*

Our unforgeability experiment captures a property known as *consistency of credentials* [16, 39] from the anonymous credentials literature. Consistency of credentials requires that a set of holders cannot collaborate to generate a valid identity proof that one holders alone cannot generate. In our unforgeability experiment, the adversary can output an identity proof $\mathtt{Proof}^*$ that uses credentials issued to several (corrupt) holders. Such a strategy can satisfy the two conditions of our unforgeability experiment. In particular, the credentials used in the identity proof are not contained in the credential list of a single corrupt holder, satisfying the second condition. If the identity proof is valid, the adversary succeeds in our unforgeability experiment. Accordingly, if unforgeability holds, the adversary cannot adopt this attack to succeed.

Anonymity. In an anonymous credential system, a verifier should be unable to determine which holder generated an identity proof, even if the verifier colludes with credential issuers. We capture this intuition in an anonymity experiment (Definition 3). In our experiment, the adversary outputs two challenge holder indices i_0^* and i_1^* and a presentation request $\mathtt{PReq}^*$, and obtains a challenge identity proof $\mathtt{Proof}^*$ on behalf of holder i_b^* for a bit b chosen randomly from the set $\{0, 1\}$ in the experiment. As in our unforgeability experiment, the adversary can query any oracles defined in Figure 2, and also has access to the oracles in Figure 4. The adversary then outputs a 'guess' at bit b and we say that the HLAC protocol satisfies anonymity if the adversary succeeds in outputting a bit $b' = b$ with probability at most negligibly greater than $1/2$. We also require that the following conditions hold. Firstly, i_0^* and i_1^* must not be queried to the oracle $\mathcal{O}\mathrm{CorruptH}$ (i.e., both challenge holders must be honest). This prevents the adversary from learning, for example, master secrets that may provide the adversary with some distinguishing advantage. Secondly, the presentation request $\mathtt{PReq}^*$ output by the adversary must be satisfiable by the credential lists of both challenge holders. Otherwise, the adversary can output a presentation request that can only be satisfied by one of the challenge holders and trivially succeed in the experiment. Finally, the set of revealed attributes $\mathtt{R}$ contained in $\mathtt{PReq}^*$ must be identical for both challenge holders. Else, the adversary can request revealed attributes in the presentation request such that the two challenge holders will reveal different values, and the adversary can trivially distinguish.

Definition 3 (Anonymity). *The* HLAC *protocol satisfies* anonymity *if, for any probabilistic polynomial time adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible*

function negl *such that*

$$\Pr\left[\begin{array}{c} b \leftarrow\$ \{0,1\};\ \top \leftarrow \mathcal{O}\mathrm{Setup}() \\ (\mathsf{st}, \mathtt{PReq}^*, i_0^*, i_1^*) \leftarrow\$ \mathcal{A}_1^{\mathcal{O}}() \\ \mathtt{Proof}^* \leftarrow\$ \mathsf{Present}(\mathtt{PReq}^*, \mathbf{L}[i_b^*], \mathbf{H}[i_b^*]) \\ b' \leftarrow\$ \mathcal{A}_2(\mathsf{st}, \mathtt{Proof}^*) \\ \textbf{return}\ \left(b'=b \ \wedge\ \{i_0^*, i_1^*\} \subseteq \mathrm{QH} \setminus \mathrm{QCH} \ \wedge\ \mathtt{R} \subseteq \mathsf{V}_0 \cap \mathsf{V}_1 \right. \\ \left. \wedge\ \{\psi(\mathtt{PReq}^*, \mathbf{L}[i_j^*]) = 1\}_{j \in \{0,1\}}\right) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}$$

where $\mathcal{O} = \{\mathcal{O}\mathrm{RegH}, \mathcal{O}\mathrm{CorruptH}, \mathcal{O}\mathrm{RegCDef}, \mathcal{O}\mathrm{CorruptCDef}, \mathcal{O}\mathrm{CredIssuance}, \mathcal{O}\mathrm{Offer}, \mathcal{O}\mathrm{Request}, \mathcal{O}\mathrm{Issue}, \mathcal{O}\mathrm{Store}, \mathcal{O}\mathrm{Proof}, \mathcal{O}\mathrm{Revoke}\}$ *are the oracles defined in Figures 2, 3 and 4, and* V_b *is the set of attribute values in* $\mathbf{L}[i_b]$ *for* $b \in \{0,1\}$.

3 Preliminaries

In this section, we introduce the building blocks for the HLAC protocol. We also state the hardness assumptions and other details that are required for the security analysis of the protocol. The HLAC protocol employs Camenisch-Lysyanskaya (CL) signatures [18], non-interactive zero-knowledge proofs described in [43], the Pedersen commitment scheme [42], and the Camenisch-Kohlweiss-Soriente (CKS) cryptographic accumulator and associated zero-knowledge proof of knowledge from [14]. HLAC also requires a hash function $\mathcal{H}\colon \mathcal{X} \to \{0,1\}^\ell$ that takes as input some element from domain $\mathcal{X}$ and outputs a string of fixed length ℓ. We assume that all building blocks and algorithms in the HLAC protocol have access to a cyclic group $\mathbb{G}$ with generators g and h of prime order q.

3.1 Assumptions and Definitions

Definition 4 (Bilinear Group). *We define a type 3 bilinear pairing as follows. Let* $G_1 = E(F_p)$ *and* $G_2 = E(F_{p^2})$ *where* E *is the BN254 curve defined over a 254-bit prime* p*. Let* $e\colon G_1 \times C_2 \to G_T$ *where* G_T *is the group of* q^{th} *roots of unity in* $F_{p^{12}}$ *where* $q_1 = |E(F_p)|$ *and* q_1 *is also a 254-bit prime. That is,* G_1 *and* G_2 *are cyclic additive groups of prime order* q_1*, and* G_t *is a multiplicative cyclic group of order* q_1 *with a pairing* e*.*

Definition 5 (Quadratic Residue). *We say that an integer* q *is a* quadratic residue $\mod n$ *if there exists an integer* x *such that* $x^2 \equiv q \mod n$*. We write that* $q \in \mathcal{QR}_n$*.*

Definition 6 (Strong RSA Assumption [3, 33]). *The* strong RSA assumption *holds if, for a random RSA modulus* n*, element* $u \in \mathbb{Z}_n$ *and any probabilistic polynomial time adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that*

$$\Pr[(v,e) \leftarrow\$ \mathcal{A}(n,u)\ :\ v^e \equiv u \mod n] \leq \mathsf{negl}.$$

Definition 7 (Discrete Logarithm Assumption). *The* discrete logarithm assumption *holds in group* $(\mathbb{G}, p, g)$ *if, for any probabilistic polynomial time adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that*

$$\Pr[y \leftarrow\!\$\, \mathbb{G};\ x \leftarrow\!\$\, \mathcal{A}(\mathbb{G}, p, g, y)\ :\ g^x \equiv y] \leq \mathsf{negl}.$$

Definition 8 (Diffie-Hellman Exponent assumption [14]). *Let* $\mathbb{G}$ *be a cyclic group with prime order* p *and generator* $g \in \mathbb{G}$*. The* Diffie-Hellman Exponent (DHE) assumption *holds if, for any probabilistic polynomial time adversary* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\mathsf{Pr}\Big[\gamma \leftarrow\!\$\, \mathbb{Z}_q;\ \{\textit{for } i \in \{1, \ldots, n, n+2, \ldots, 2n\}\ :\ g_i = g^{\gamma^i}\}\ :\ g_{n+1} \leftarrow \mathcal{A}(\{g, g_1, \ldots, g_n, g_{n+2}, \ldots, g_{2n}\})\Big] \leq \mathsf{negl}.$$

Definition 9 (Hidden Strong Diffie-Hellman Exponent assumption [14]). *Let* $\mathbb{G}$ *be a cyclic group with prime order* p *and generator* $g \in \mathbb{G}$*. The* Hidden Strong Diffie-Hellman Exponent (HSDHE) assumption *holds if, for any probabilistic polynomial time adversary* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\mathsf{Pr}\Big[g, g^x, u \leftarrow\!\$\, \mathbb{G};\ \{\textit{for } i \in [n]\ :\ g^{1/(x+\gamma^i)}, g^{\gamma^i}, u^{\gamma^i}\};\ \{\textit{for } i \in [n+2, 2n]\ :\ g^{\gamma^i},\}\ (g^{1/(x+c)}, g^c, u^c) \leftarrow \mathcal{A}(g, g^x, u, g^{1/(x+\gamma^i)}, g^{\gamma^i}, u^{\gamma^i}, g^{\gamma^i})\Big] \leq \mathsf{negl}.$$

3.2 Camenisch-Lysyanskaya Signature Scheme [18]

The CL signature scheme is used to generate a signature over all attribute values in a credential during the credential issuance phase. The Camenisch-Lysyanskaya (CL) signature scheme CL [18] is typically defined as a tuple of algorithms $(\mathsf{CL.KGen}, \mathsf{CL.Sign}, \mathsf{CL.Verify})$. However, in the HLAC construction, algorithm $\mathsf{CL.Verify}$ is not used. Therefore, we define the CL signature scheme in Figure 5 as a pair of algorithms $(\mathsf{CL.KGen}, \mathsf{CL.Sign})$. Algorithm $\mathsf{CL.KGen}$ outputs a public/private key pair $(\mathsf{ppk}, \mathsf{psk})$ and key metadata pmd. Algorithm $\mathsf{CL.Sign}$ produces a signature σ over a set of messages $m_1, \ldots, m_L$ (i.e., the attribute values in the HLAC protocol) and outputs signature σ and some signing metadata Q. Rather than define algorithm $\mathsf{CL.Verify}$, the HLAC protocol requires generation of a proof of correct signing, verification of which includes verification of the signature. The private key and signing metadata are used to generate a zero-knowledge proof of correct signing.

For the security of the HLAC construction, we require that the CL signature scheme satisfies security against adaptive chosen message attacks.

Definition 10 (Security against adaptive chosen message attacks [34]). *A signature scheme satisfies* security against adaptive chosen message attacks

$\mathsf{CL.KGen}(L)$

$p \leftarrow 2p' + 1$ s.t. $p, p' \in \mathbb{P} \ \wedge \ p' \leftarrow\$ \{0,1\}^{1536}$
$q \leftarrow 2q' + 1$ s.t. $q, q' \in \mathbb{P} \ \wedge \ q' \leftarrow\$ \{0,1\}^{1536}$
$n \leftarrow p \cdot q$
$S \leftarrow\$ \mathcal{QR}_n$
for $i \in 1, \ldots, L$
  $x_i \leftarrow\$ [2, p'q' - 1]$
  $R_i \leftarrow S^{x_i} \mod n$
$x_z \leftarrow\$ [2, p'q' - 1]$
$Z \leftarrow S^{x_z} \mod n$
$\mathsf{psk} \leftarrow (p, q)$
$\mathsf{ppk} \leftarrow (n, S, Z, \{R_0, \ldots, R_L\})$
$\mathsf{pmd} = (x_z, x_1, \ldots, x_L)$
return $(\mathsf{ppk}, \mathsf{psk}, \mathsf{pmd})$

$\mathsf{CL.Sign}(\mathsf{ppk}, \mathsf{psk}, \{v_1, \ldots, v_L\})$

parse ppk as $(n, S, Z, \{R_0, \ldots, R_L\})$
parse psk as (p, q)
$v'' \leftarrow\$ \{0,1\}^{2724}$
$e \leftarrow\$ \left[2^{596}, 2^{596} + 2^{119}\right]$ s.t. $e \in \mathbb{P}$
$Q \leftarrow \dfrac{Z}{v_1 \cdot S^{v''} \prod_{[2,L]} R_i^{v_i} \mod n} \mod n$
$A \leftarrow Q^{(e^{-1} \mod p'q')} \mod n$
$\sigma \leftarrow (A, e, v'')$
return (σ, Q)

$\mathsf{ZK}_\sigma.\mathsf{Prove}((\sigma, n_1), (Q, \mathsf{psk}))$

parse σ as (A, e, v'')
parse psk as (p, q)
$r \leftarrow\$ \mathbb{Z}_{p'q'}$
$\tilde{A} \leftarrow Q^r \mod n$
$c_\sigma \leftarrow \mathcal{H}(Q \| A \| \tilde{A} \| n_1)$
$s_e \leftarrow r - c_\sigma \cdot e^{-1} \mod p'q'$
return $\rho_\sigma \leftarrow (s_e, c_\sigma)$

$\mathsf{ZK}_\sigma.\mathsf{Verify}((\sigma, n_1), \mathsf{ppk}, (\mathsf{sReq}, \{v_1, \ldots, v_L\}), \rho_\sigma)$

parse ppk as $(n, S, Z, \{R_1, \ldots, R_L\})$
parse σ as (A, e, v'')
parse ρ_σ as (s_e, c_σ)
if $e \notin \left[2^{596}, 2^{596} + 2^{119}\right] \vee e \notin \mathbb{P}$ **return** 0
$Q' \leftarrow \dfrac{Z}{S^v \prod_{i \in [L]} R_i^{v_i}} \mod n$
if $Q' \neq A^e \mod n$ **return** 0
$\tilde{A}' \leftarrow A^{c_\sigma + s_e \cdot e} \mod n$
if $c_\sigma \neq \mathcal{H}(Q' \| A \| \tilde{A}' \| n_1)$ **return** 0
return 1

Fig. 5: The CL signature scheme ($\mathsf{CL.KGen}, \mathsf{CL.Sign}$) and the NIZK proof system for correct key construction ($\mathsf{ZK}_\sigma.\mathsf{Prove}, \mathsf{ZK}_\sigma.\mathsf{Verify}$).

if, for any probabilistic polynomial time adversary $\mathcal{A}$, there exists a negligible function negl *such that*

$$\Pr\left[\begin{array}{c}\mathcal{Q}\leftarrow\emptyset;\ (\mathsf{ppk},\mathsf{psk},\mathsf{pmd})\leftarrow\$\ \mathsf{CL.KGen}() \\ (m^*=(m_1,\ldots,m_L),\sigma^*=(A,e,v'')) \leftarrow\$\ \mathcal{A}^{\mathcal{O}\mathrm{Sign}}(\mathsf{ppk}) \\ \textbf{return}\ \left(A^e=\frac{Z}{S^{v''}\prod_{i\in[L]}R_i^{m_i}}\right. \\ \left.\wedge\ m^*\notin\mathcal{Q}\right)\end{array}\right]\leq\mathsf{negl}$$

where $\mathcal{O}\mathrm{Sign}(m)$ *computes and outputs* $\sigma\leftarrow\mathsf{CL.Sign}(\mathsf{ppk},\mathsf{psk},m)$*, and updates set* $\mathcal{Q}$ *to include message* m*.*

Theorem 1. *The CL signature scheme satisfies security against adaptive chosen message attacks if the strong RSA assumption holds [18].*

3.3 Pedersen Commitments [42]

As mentioned in Section 2.1, the first attribute of every issued credential is a blinded version of the holder's master secret. In the HLAC protocol, the master secret is blinded during credential issuance by constructing a Pedersen commitment to the master secret. The Pedersen commitment scheme Comm is a tuple of algorithms $(\mathsf{KGen},\mathsf{Commit},\mathsf{Open})$ defined as follows.

Key Generation Choose a commitment key $h\leftarrow\$\,\mathbb{G}$ where $\mathbb{G}$ is a cyclic group with prime order p with generator $g\in\mathbb{G}$.

Commit To commit to a message m, choose $r\leftarrow\$\,\mathbb{Z}_p$ and return the commitment $c=g^r\cdot h^m$.

Open On input a message m, commitment c and randomness r, output 1 if c commits to message m, and 0 otherwise.

Security of the HLAC protocol requires that the Pedersen commitment scheme satisfies binding and hiding.

Definition 11 (Binding [42]). *The Pedersen commitment scheme satisfies* binding *if, for any probabilistic polynomial time adversary $\mathcal{A}$, there exists a negligible function* negl *such that*

$$\Pr\left[\begin{array}{c}h\leftarrow\$\,\mathsf{Comm.KGen}();\ (m_0,m_1,r_0,r_1)\leftarrow\$\,\mathcal{A}(h) \\ \textbf{return}\ \left(\mathsf{Commit}(h,m_0;r_0)=\mathsf{Commit}(h,m_1;r_1)\right. \\ \left.\wedge\ m_0\neq m_1\right)\end{array}\right]\leq\mathsf{negl}.$$

Definition 12 (Hiding [42]). *The Pedersen commitment scheme satisfies* hiding *if, for any probabilistic polynomial time adversary $\mathcal{A}$, there exists a negligible function* negl *such that*

$$\Pr\left[\begin{array}{c}b\leftarrow\$\,\{0,1\};\ h\leftarrow\$\,\mathsf{Comm.KGen}();\ (m_0,m_1,\mathsf{st})\leftarrow\$\,\mathcal{A}_1(h) \\ c\leftarrow\$\,\mathsf{Commit}(h,m_b);\ b'\leftarrow\$\,\mathcal{A}_2(\mathsf{st},c) \\ \textbf{return}\ b'=b\end{array}\right]\leq\frac{1}{2}+\mathsf{negl}.$$

Theorem 2. *The Pedersen commitment scheme is perfectly hiding and computationally binding if the discrete logarithm assumption holds with respect to the cyclic group $\mathbb{G}$ of prime order p with group generator g [42].*

3.4 Cryptographic Accumulator [14]

Cryptographic accumulators generate a hash of a list of values (an accumulator) and a witness that can be used to demonstrate that a value is included in the accumulator. They are used in anonymous credential systems to support revocation of credentials. Indeed, issuers can compute an accumulator that contains values associated with each non-revoked credential to issue revocable credentials, and holders can use a witness generated during credential issuance (and included in their credential) to demonstrate that the value associated with their credential is included in the accumulator. In this way, the holder can demonstrate that their credential is not revoked.

The HLAC protocol uses a modified version of the CKS cryptographic accumulator [14] that is defined as a tuple of algorithms (RevKGen, AccKGen, AccAdd, AccUpd, Acc.WUpd), as in Figure 6. In particular, the HLAC protocol uses type-3 bilinear pairings, rather than type-1 pairings used in the original construction. Accordingly, we assume that each algorithm in the HLAC protocol and the CKS protocol has access to a set of public parameters $\mathsf{pp} = (G_1, G_2, G_T, e, q_1, g, g')$ where G_1, G_2 and G_T are groups of prime order q_1, e is a map such that $e\colon G_1 \times G_2 \to G_T$, g is a generator of G_1 and g' is a generator of G_2. Algorithms RevKGen and AccKGen are used during setup to generate keys and initial revocation lists. When a new credential is issued, algorithms AccAdd and AccUpd are run to update the cryptographic accumulator and revocation list, and to generate a witness for the credential. Finally, algorithm Acc.WUpd is used by the holder to update their witness when generating an identity proof, in order to demonstrate that their credential is included in the most recent accumulator generated by the issuer and posted to the ledger.

3.5 Zero-Knowledge Proofs [14, 43]

Non-interactive zero-knowledge proofs of knowledge (NIZKs) are used in the HLAC protocol to generate identity proofs, to prove correct issuer key construction during setup, and to prove correct blinding of the master secret and correct credential signing during credential issuance. For these purposes, the HLAC protocol uses NIZK proofs described in [43].

Generically, a proof system ZK is a protocol between a prover and a verifier where the prover attempts to prove that a statement s is in some language $\mathcal{L}$. To achieve this, the prover demonstrates knowledge of a witness w such that the tuple $(s, w) \in \mathcal{R}$ for some binary relation $\mathcal{R}$. The proof systems used in the HLAC protocol are all Σ-protocols, which are interactive protocols wherein the prover sends a commitment to a verifier, who generates a challenge. The prover then sends a response to the challenger. Σ-protocols can be transformed into non-interactive zero-knowledge (NIZK) proof system using the Fiat-Shamir transform [27], which replaces the verifier's challenge with the output of a random oracle, usually instantiated with a hash function, on input of the protocol transcript. We define NIZK proof systems to be a pair of algorithms (ZK.Prove, ZK.Verify) where ZK.Prove takes as input a statement s and witness

RevKGen(pp)

parse pp as $(G_1, G_2, G_T, e, q_1, g, g')$
$h, h_0, h_1, h_2, \tilde{h} \leftarrow\$ (G_1)^5$
$sk \leftarrow\$ \mathbb{Z}_{q_1}$
$pk \leftarrow g^{sk}$
$u, \hat{h} \leftarrow\$ (G_2)^2$
$x \leftarrow\$ \mathbb{Z}_{q_1}$
$y \leftarrow \hat{h}^x$
$\mathsf{rpk} \leftarrow (\mathsf{pp}, g, g', h, h_0, h_1, h_2, \tilde{h}, \hat{h}, u, pk, y)$
$\mathsf{rsk} \leftarrow (x, sk)$
return $(\mathsf{rpk}, \mathsf{rsk})$

AccKGen(pp, max)

parse pp as $(G_1, G_2, G_T, e, q_1, g, g')$
$\mathsf{ask} \leftarrow\$ \mathbb{Z}_{q_1}$
$\mathsf{apk} \leftarrow (e(g, g'))^{\mathsf{ask}^{\mathtt{max}+1}}$
$\mathsf{T} \leftarrow ()$
for $i \in 1, \ldots, 2 \cdot \mathtt{max}$
  $\mathsf{T}[i] \leftarrow g'^{\mathsf{ask}^i}$
$\mathsf{T}[\mathtt{max}+1] \leftarrow g'^{\mathsf{ask}}$
$\mathsf{Acc} \leftarrow 0$
$\mathsf{RL} \leftarrow \mathbf{1}$
return $(\mathsf{apk}, \mathsf{ask}, \mathsf{T}, \mathsf{Acc}, \mathsf{RL})$

AccAdd(rpk, rsk, ask, max, i, RL, m_2, u_r, pp)

parse pp as $(G_1, G_2, G_T, e, q_1, g, g')$
parse rpk as $(\mathsf{pp}, g, g', h, h_0, h_1, h_2, \tilde{h}, \hat{h}, u, pk, y)$
parse rsk as (x, sk)
$c, v_r'' \leftarrow\$ (G_2)^2 \mod q_1$
$g_i \leftarrow g^{\mathsf{ask}^i}$
$\sigma \leftarrow (h_0 \cdot h_1^{\mathsf{m}_2} \cdot u_r \cdot g_i \cdot h_2^{v_r''})^{\frac{1}{x+c}}$
$\sigma_i \leftarrow g'^{\frac{1}{sk+\mathsf{ask}^i}}$
$u_i \leftarrow u^{\mathsf{ask}^i}$
$\mathsf{w} \leftarrow \prod_{j\,:\,\mathsf{RL}[j]=0}^{j \neq i} g'_{\mathtt{max}+1-j+i}$
$\mathsf{w}_\sigma \leftarrow (\sigma_i, u_i, g_i)$
$\sigma_r \leftarrow (\sigma, c, v_r'', \mathsf{w}_\sigma, g_i, i, \mathsf{m}_2)$**return** (σ_r, w)

AccUpd(Acc, max, $\mathtt{C}_\mathtt{I}$, $\mathtt{C}_\mathtt{R}$, RL, ask)

$\mathsf{pw} \leftarrow 0$
$\forall\ i \in \mathtt{C}_\mathtt{I}\ :\ \mathsf{pw} \leftarrow \mathsf{pw} + \mathsf{ask}^{\mathtt{max}+1-i}$
$\forall\ i \in \mathtt{C}_\mathtt{R}\ :\ \mathsf{pw} \leftarrow \mathsf{pw} - \mathsf{ask}^{\mathtt{max}+1-i}$
$\mathsf{Acc}_{\mathsf{new}} \leftarrow g'^{\mathsf{pw}} \cdot \mathsf{Acc}$
$\forall\ i \in \mathtt{C}_\mathtt{I}\ :\ \mathsf{RL}[i] \leftarrow 0$
$\forall\ i \in \mathtt{C}_\mathtt{R}\ :\ \mathsf{RL}[i] \leftarrow 1$
return $(\mathsf{Acc}_{\mathsf{new}}, \mathsf{RL})$

Acc.WUpd(w, RL, RL′, T)

$$a = \frac{\prod_{j \in \mathsf{RL}' \setminus \mathsf{RL}} \mathsf{T}[\mathtt{max}+1-j+i]}{\prod_{j \in \mathsf{RL} \setminus \mathsf{RL}'} \mathsf{T}[\mathtt{max}+1-j+i]}$$

$\mathsf{w}' = \mathsf{w} \cdot a$
return w'

Fig. 6: The CKS cryptographic accumulator (RevKGen, AccKGen, AccAdd, AccUpd, Acc.WUpd).

w and outputs a proof ρ, and $\mathsf{ZK.Verify}$ takes as input s and ρ and outputs a bit that is 1 if the proof verifies and 0 otherwise.

For our security analysis, we recall definitions of soundness and zero-knowledge for a NIZK proof system from [35]. Intuitively, *soundness* states that an adversary cannot output a verifiable proof unless there exists an algorithm that can extract a witness w, and *zero-knowledge* is the property that a NIZK proof reveals nothing about the witness. More formally, we recall the following definitions from [35].

Definition 13 (Soundness [35]). *A NIZK proof system satisfies* soundness *if, for all polynomial time adversaries $\mathcal{A}$, there exists a negligible function* negl *such that*

$$\Pr\Big[(s,\rho) \leftarrow\$ \mathcal{A}() \ : \ \begin{smallmatrix}(s,w) \ \in \ \mathcal{R} \\ \vee \ \mathsf{ZK.Verify}(s,\rho) \ \coloneqq \ 0\end{smallmatrix}\Big] \geq 1 - \mathsf{negl}.$$

Definition 14 (Zero-knowledge [35]). *A NIZK proof system satisfies* zero-knowledge *if, for all probabilistic polynomial time adversaries $\mathcal{A}$, there exists a negligible function* negl *such that,*

$$\Big|\Pr\big[\mathcal{A}^{\mathcal{O}\mathrm{Prove}}() = 1\big] - \Pr[\mathcal{A}^{\mathcal{O}\mathrm{Sim}}()=1]\Big| \leq \mathsf{negl}$$

where $\mathcal{O}\mathrm{Prove}(s,w)$ returns $\mathsf{ZK.Prove}(s,w)$ and $\mathcal{O}\mathrm{Sim}(s,w)$ returns $\mathsf{ZK.Prove}(s,\tau)$ on input of query $(s,w) \in \mathcal{R}$, for some trapdoor τ output by a simulated setup.

Definition 15 (Secure NIZK Proof System). *A NIZK proof system is* secure *if it satisfies soundness and zero-knowledge.*

We now formally define the non-interactive zero-knowledge proof systems used in the HLAC protocol, and note that the NIZK protocols presented here are all secure. We provide references to formal proofs and further details on these proofs where relevant.

Correct Key Construction. Algorithm DSetup outputs a NIZK proof of correct key construction for the CL signature key pair $\mathsf{ppk} = (n, S, \{R_1, \ldots, R_L\}, Z)$ and $\mathsf{psk} = (p, q)$, we define a NIZK proof system $(\mathsf{ZK}_{\mathsf{ppk}}.\mathsf{Prove}, \mathsf{ZK}_{\mathsf{ppk}}.\mathsf{Verify})$ in Figure 7. The NIZK proof system $(\mathsf{ZK}_{\mathsf{ppk}}.\mathsf{Prove}, \mathsf{ZK}_{\mathsf{ppk}}.\mathsf{Verify})$ is secure if Z and $\{R_i\}_{i\in[L]}$ are elements of the subgroup generated by S [43].

Blinding of the Master Secret. To request a credential, the holder must blind their master secret and generate a zero-knowledge proof of correct blinding. To facilitate this, we define NIZK proof system $(\mathsf{ZK}_{\mathtt{ms}}.\mathsf{Prove}, \mathsf{ZK}_{\mathtt{ms}}.\mathsf{Verify})$ in Figure 8. The NIZK proof system $(\mathsf{ZK}_{\mathtt{ms}}.\mathsf{Prove}, \mathsf{ZK}_{\mathtt{ms}}.\mathsf{Verify})$ is a secure NIZK proof for equality of discrete logarithms [43].

$\mathsf{ZK}_{\mathsf{ppk}}.\mathsf{Prove}((\mathsf{ppk}, \{\mathtt{a}_1, \ldots, \mathtt{a}_L\}), (\mathsf{psk}, \mathsf{pmd}))$

parse ppk as $(n, S, Z, \{R_1, \ldots, R_L\})$
parse pmd as $(x_z, x_1, \ldots, x_L)$
parse psk as (p, q)
$\alpha_z \leftarrow\!\!\$\, [2, p'q' - 1]$
$\tilde{Z} \leftarrow S^{\alpha_z}$
for $i \in 1, \ldots, L$
  $\alpha_i \leftarrow\!\!\$\, [2, p'q' - 1]$
  $\tilde{R}_i \leftarrow S^{\alpha_i}$
$c_{\mathsf{ppk}} \leftarrow \mathcal{H}(Z \| \{R_1, \ldots, R_L\} \| \tilde{Z} \| \{\tilde{R}_1, \ldots \tilde{R}_L\})$
$\tilde{x}_z \leftarrow \alpha_z + c_{\mathsf{ppk}} \cdot x_z \mod p'q'$
for $i \in 1, \ldots, L$
  $\tilde{x}_i \leftarrow \alpha_i + c_{\mathsf{ppk}} \cdot x_i \mod p'q'$
return $\rho_{\mathtt{pDef}} = (c_{\mathsf{ppk}}, \tilde{x}_z, \{(\tilde{x}_i, \mathtt{a}_i)\}_{i \in [L]})$

$\mathsf{ZK}_{\mathsf{ppk}}.\mathsf{Verify}((\mathsf{ppk}, \{\mathtt{a}_1, \ldots, \mathtt{a}_L\}), \rho_{\mathtt{pDef}})$

parse ppk as $(n, S, Z, \{R_1, \ldots, R_L\})$
parse $\rho_{\mathtt{pDef}}$ as $(c_{\mathsf{ppk}}, \tilde{x}_z, \{(\tilde{x}_i, \mathtt{a}'_i)\}_{i \in [L]})$
if $\{\mathtt{a}_1, \ldots, \mathtt{a}_L\} \neq \{\mathtt{a}'_1, \ldots, \mathtt{a}'_L\}$ **return** 0
for $i \in 1, \ldots, L$
  $\tilde{R}'_i \leftarrow R_i^{-c_{\mathsf{ppk}}} S^{\tilde{x}_i}$ s.t. $R_i^{-1} R_i = 1 \mod n$
$\tilde{Z}' \leftarrow Z^{-c_{\mathsf{ppk}}} S^{\tilde{x}_z}$ s.t. $Z^{-1} Z = 1 \mod n$
if $c_{\mathsf{ppk}} \neq H(Z \| \{R_1, \ldots, R_L\} \| \tilde{Z}' \| \{\tilde{R}'_i\}_{i \in [L]})$ **return** 0
return 1

Fig. 7: The NIZK proof system $(\mathsf{ZK}_{\mathsf{ppk}}.\mathsf{Prove}, \mathsf{ZK}_{\mathsf{ppk}}.\mathsf{Verify})$.

$\mathsf{ZK}_{\mathtt{ms}}.\mathsf{Prove}((\mathsf{ppk}, \mathsf{U}, n_0), (v', \mathtt{ms}))$

parse ppk as $(n, S, Z, \{R_1, \ldots, R_L\})$
$\tilde{v} \leftarrow\$ \{0,1\}^{3488}$
$\tilde{\mathtt{ms}} \leftarrow\$ \{0,1\}^{593}$
$\tilde{\mathsf{U}} \leftarrow (S^{\tilde{v}}) R_1^{\tilde{\mathtt{ms}}} \mod n$
$c_{\mathsf{U}} \leftarrow \mathcal{H}(\mathsf{U} \| \tilde{\mathsf{U}} \| n_0)$
$\tilde{v}' \leftarrow \tilde{v} + c_{\mathsf{U}} \cdot v'$
$\tilde{\mathtt{ms}}' \leftarrow \tilde{\mathtt{ms}} + c_{\mathsf{U}} \cdot \mathtt{ms}$
return $\rho_{\mathsf{U}} \leftarrow (c_{\mathsf{U}}, \tilde{v}', \tilde{\mathtt{ms}}')$

$\mathsf{ZK}_{\mathtt{ms}}.\mathsf{Verify}((\mathsf{ppk}, \mathsf{U}, n_0), \rho_{\mathsf{U}})$

parse ppk as $(n, S, Z, \{R_1, \ldots, R_L\})$
parse ρ_{U} as $(c_{\mathsf{U}}, \tilde{v}', \tilde{\mathtt{ms}}')$
$\tilde{\mathsf{U}}' \leftarrow \mathsf{U}^{-c_{\mathsf{U}}} \cdot R_1^{\tilde{\mathtt{ms}}'} \cdot S^{\tilde{v}'}$ s.t. $\mathsf{U}^{-1}\mathsf{U} = 1 \mod n$
if $c_{\mathsf{U}} \neq \mathcal{H}(\mathsf{U} \| \tilde{\mathsf{U}}' \| n_0)$ **return** 0
return 1

Fig. 8: The NIZK proof system $(\mathsf{ZK}_{\mathtt{ms}}.\mathsf{Prove}, \mathsf{ZK}_{\mathtt{ms}}.\mathsf{Verify})$.

Correct Credential Signing. To generate a credential, algorithm Issue generates a NIZK proof of correct signing using NIZK proof system $(\mathsf{ZK}_{\sigma}.\mathsf{Prove}, \mathsf{ZK}_{\sigma}.\mathsf{Verify})$, defined in Figure 5. This NIZK protocol is an example of Schnorr's signature scheme, which is a secure NIZK proof system [43] in the random oracle model [6]. The zero-knowledge verification algorithms also acts as a CL signature verification algorithm.

Credential Issuance. To generate an identity proof, algorithm $\mathsf{Present}$ generates a NIZK proof that each credential used in the identity proof was signed by an issuer. We define NIZK proof system $(\mathsf{ZK}_{\mathtt{cred}}.\mathsf{Prove1}, \mathsf{ZK}_{\mathtt{cred}}.\mathsf{Prove2}, \mathsf{ZK}_{\mathtt{cred}}.\mathsf{Verify})$ in Figure 9. The above protocol is a secure NIZK proof [43] as it is a variation of the Schnorr protocol modulo a composite.

Predicate Proofs. To generate an identity proof, algorithm $\mathsf{Present}$ also generates a NIZK proof for each predicate. We define NIZK proof system $(\mathsf{ZK}_{\mathsf{pred}}.\mathsf{Prove1}, \mathsf{ZK}_{\mathsf{pred}}.\mathsf{Prove2}, \mathsf{ZK}_{\mathsf{pred}}.\mathsf{Verify})$ in Figure 10. As expected, the above NIZK proof protocol is a secure NIZK proof [43].

Non-Revocation Proof. In addition to a credential proof and predicate proof, holders must present a non-revocation proof to demonstrate that their credential is currently active. We present the NIZK proof system $(\mathsf{ZK}_{\mathsf{rev}}.\mathsf{Prove1}, \mathsf{ZK}_{\mathsf{rev}}.\mathsf{Prove2}, \mathsf{ZK}_{\mathsf{rev}}.\mathsf{Verify})$ from [14] in Figures 11 and 12. This system is secure if the n-DHE and n-HSDHE assumptions hold [14].

$\mathsf{ZK}_{\mathtt{cred}}.\mathsf{Prove1}((A_{\bar{r}}, \mathsf{ppk}), (\sigma, \{\hat{v}_j\}_{j \in A_{\bar{r}}}))$

parse ppk as $(n, S, Z, \{R_1, \ldots, R_L\})$
parse σ as (A, e, v'')
$r \leftarrow\$ \{0,1\}^{3152}$
$\bar{A} \leftarrow A \cdot S^r \mod n$
$\bar{v} \leftarrow v'' - e \cdot r$
$\mathsf{C}_{\mathtt{cred}} \leftarrow (\bar{A}, \bar{v})$
$\bar{e} \leftarrow e - 2^{596}$
$\bar{e}' \leftarrow\$ \{0,1\}^{456}$
$\bar{v}' \leftarrow\$ \{0,1\}^{3748}$
$\bar{Z} \leftarrow (\bar{A})^{\bar{e}'} \cdot (S)^{\bar{v}'} \cdot (\prod_{j \in A_{\bar{r}}} R_j^{\hat{v}_j})$
$\mathsf{T}_{\mathtt{cred}} \leftarrow (\bar{e}, \bar{e}', \bar{v}', \bar{Z})$
return $(\mathsf{C}_{\mathtt{cred}}, \mathsf{T}_{\mathtt{cred}})$

$\mathsf{ZK}_{\mathtt{cred}}.\mathsf{Prove2}((\mathsf{C}_{\mathtt{cred}}, A_{\bar{r}}, c_H), (\mathsf{T}_{\mathtt{cred}}, \{\hat{v}_j, v_j\}_{j \in A_{\bar{r}}})$

parse $\mathsf{C}_{\mathtt{cred}}$ as $(\bar{A}, \bar{v})$
parse $\mathsf{T}_{\mathtt{cred}}$ as $(\bar{e}, \bar{e}', \bar{v}', \bar{Z})$
$\tilde{e} \leftarrow \bar{e}' + c_H \cdot \bar{e}$
$\tilde{v} \leftarrow \bar{v}' + c_H \cdot \bar{v}$
$\forall\ j \in 1, \ldots, |A_{\bar{r}}|$
  $\tilde{v}_j \leftarrow \hat{v}_j + c_H \cdot v_j$
$\tilde{\mathsf{m}_2} \leftarrow \hat{v}_2 + c_H \cdot \mathsf{m}_2$
return $\rho_{\mathtt{cred}} \leftarrow (\bar{A}, \tilde{e}, \tilde{v}, \{\tilde{v}_j\}_{j \in A_{\bar{r}}}, \tilde{\mathsf{m}_2})$

$\mathsf{ZK}_{\mathtt{cred}}.\mathsf{Verify}((A_{\bar{r}}, A_r, c_H, \{v_j\}_{j \in A_r}), \mathsf{ppk}, \rho_{\mathtt{cred}})$

parse $\rho_{\mathtt{cred}}$ as $(\bar{A}, \tilde{e}, \tilde{v}, \{\tilde{v}_j\}_{j \in A_{\bar{r}}}, \tilde{\mathsf{m}_2})$
parse ppk as $(n, S, Z, \{R_1, \ldots, R_L\})$
$\hat{T} \leftarrow \left(\frac{Z}{(\prod_{j \in A_r} R_j^{v_j})(\bar{A})^{2^{596}}}\right)^{-c_H} \cdot (\bar{A})^{\tilde{e}} \left(\prod_{j \in A_{\bar{r}}} R_j^{\tilde{v}_j}\right) (S^{\tilde{v}}) \mod n$
$\hat{\mathsf{T}}_{\mathtt{cred}} \leftarrow \hat{T}$
return $\hat{\mathsf{T}}_{\mathtt{cred}}$

Fig. 9: The NIZK proof system $(\mathsf{ZK}_{\mathtt{cred}}.\mathsf{Prove1}, \mathsf{ZK}_{\mathtt{cred}}.\mathsf{Prove2}, \mathsf{ZK}_{\mathtt{cred}}.\mathsf{Verify})$.

$\mathsf{ZK}_{\mathsf{pred}}.\mathsf{Prove1}((\mathsf{ppk}, p, z), (v, \hat{v}))$

parse ppk as $(n, S, Z, \{R_1, \ldots, R_L\})$

$$\Delta \leftarrow \begin{array}{lll} z - v & \textbf{if} & p = \leq \\ z - v - 1 & & p = < \\ v - z & & p = \geq \\ v - z - 1 & & p = > \end{array}$$

$$a \leftarrow \begin{array}{lll} -1 & \textbf{if} & p = \leq \text{ or } < \\ 1 & & p = \geq \text{ or } > \end{array}$$

Find u_1, u_2, u_3, u_4 s.t.

$\Delta = u_1^2 + u_2^2 + u_3^2 + u_4^2$

$r_1, r_2, r_3, r_4, r_\Delta \leftarrow\$ (\{0,1\}^{2128})^5$

for $k = 1, \ldots, 4$

$T_k \leftarrow Z^{u_k} \cdot S^{r_k} \mod n$

$T_\Delta \leftarrow Z^{\Delta} \cdot S^{r_\Delta} \mod n$

$\mathsf{C}_{\mathsf{pred}} \leftarrow (T_1, T_2, T_3, T_4, T_\Delta)$

$\tilde{\alpha} \leftarrow\$ \{0,1\}^{2787}$

$\tilde{u}_1, \tilde{u}_2, \tilde{u}_3, \tilde{u}_4 \leftarrow\$ (\{0,1\}^{592})^4$

$\tilde{r}_1, \tilde{r}_2, \tilde{r}_3, \tilde{r}_4, \tilde{r}_\Delta \leftarrow\$ (\{0,1\}^{672})^5$

$$Q \leftarrow (S^{\tilde{\alpha}}) \cdot \prod_{k=1}^{4} T_k^{\tilde{u}_k} \mod n$$

for $k = 1, \ldots, 4$

$\tilde{T}_k \leftarrow Z^{\tilde{u}_k} \cdot S^{\tilde{r}_k} \mod n$

$\tilde{T}_\Delta \leftarrow Z^{\hat{v}} \cdot (S^{a \cdot \tilde{r}_\Delta}) \mod n$

$\mathsf{T}_{\mathsf{pred}} \leftarrow (\tilde{T}_1, \tilde{T}_2, \tilde{T}_3, \tilde{T}_4, \tilde{T}_\Delta, Q)$

$\mathsf{M}_{\mathsf{pred}} \leftarrow (\{u_k, \tilde{u}_k, r_k, \tilde{r}_k\}_{k\in[4]}, r_\Delta, \tilde{r}_\Delta, \tilde{\alpha})$

return $(\mathsf{C}_{\mathsf{pred}}, \mathsf{T}_{\mathsf{pred}}, \mathsf{M}_{\mathsf{pred}})$

$\mathsf{ZK}_{\mathsf{pred}}.\mathsf{Prove2}(c_H, \mathsf{M}_{\mathsf{pred}})$

parse $\mathsf{M}_{\mathsf{pred}}$ as $(\{u_k, \tilde{u}_k, r_k, \tilde{r}_k\}_{k\in[4]}, r_\Delta, \tilde{r}_\Delta, \tilde{\alpha})$

$\hat{r}_\Delta \leftarrow \tilde{r}_\Delta + c_H \cdot r_\Delta$

for $k = 1, \ldots, 4$

$\hat{u}_k \leftarrow \tilde{u}_k + c_H \cdot u_k$

$\hat{r}_k \leftarrow \tilde{r}_k + c_H \cdot r_k$

$$\hat{\alpha} \leftarrow \tilde{\alpha} + c_H \cdot (r_\Delta - \sum_{k=1}^{4} u_k \cdot r_k)$$

return $\rho_{\mathsf{pred}} \leftarrow (\{\hat{u}_k\}_{k\in[4]}, \{\hat{r}_k\}_{k\in[4]}, \hat{r}_\Delta, \hat{\alpha})$

$\mathsf{ZK}_{\mathsf{pred}}.\mathsf{Verify}((\mathsf{ppk}, p, z), \mathsf{C}_{\mathsf{pred}}, c_H), \rho_{\mathsf{pred}}, \tilde{v})$

parse ppk as $(n, S, Z, \{R_1, \ldots, R_L\})$

parse $\mathsf{C}_{\mathsf{pred}}$ as $(T_1, T_2, T_3, T_4, T_\Delta)$

parse ρ_{pred} as $(\{\hat{u}_k\}_{k\in[4]}, \{\hat{r}_k\}_{k\in[4]}, \hat{r}_\Delta, \hat{\alpha})$

$$\Delta' \leftarrow \begin{array}{lll} z & \textbf{if} & p = \leq \text{ or } \geq \\ z - 1 & & p = < \\ z + 1 & & p = > \end{array}$$

$$a \leftarrow \begin{array}{lll} -1 & \textbf{if} & p = \leq \text{ or } < \\ 1 & & p = \geq \text{ or } > \end{array}$$

for $k = 1, \ldots, 4$

$\hat{T}_k \leftarrow T_k^{-c_H} Z^{\hat{u}_k} \cdot S^{\hat{r}_k} \mod n$

$\hat{T}_\Delta \leftarrow \left(T_\Delta^a \cdot Z^{\Delta'}\right)^{-c_H} \cdot Z^{\tilde{v}} \cdot (S^{a \cdot \hat{r}_\Delta}) \mod n$

$$\hat{Q} \leftarrow (T_\Delta)^{-c_H} \cdot \left(\prod_{k=1}^{4} T_k^{\hat{u}_k}\right) \cdot (S^{\hat{\alpha}}) \mod n$$

$\hat{\mathsf{T}}_{\mathsf{pred}} \leftarrow (\{\hat{T}_k\}_{k\in[4]}, \hat{T}_\Delta, \hat{Q})$

return $\hat{\mathsf{T}}_{\mathsf{pred}}$

Fig. 10: The NIZK proof system $(\mathsf{ZK}_{\mathsf{pred}}.\mathsf{Prove1}, \mathsf{ZK}_{\mathsf{pred}}.\mathsf{Prove2}, \mathsf{ZK}_{\mathsf{pred}}.\mathsf{Verify})$.

$\mathsf{ZK}_{\mathsf{rev}}.\mathsf{Prove1}(\mathsf{r}_{\mathtt{cred}}, \mathsf{pp}, \mathsf{rpk}, \mathsf{Acc})$

parse pp as $(G_1, G_2, G_T, e, q_1, g, g')$

parse rpk as $(\mathsf{pp}, g, g', h, h_0, h_1, h_2, \tilde{h}, \hat{h}, u, pk, y)$

parse $\mathsf{r}_{\mathtt{cred}}$ as $(\sigma, c, v_r'', \mathsf{w}_\sigma, g_i, i, \mathsf{m}_2)$

parse w_σ as (σ_i, u_i, g_i)

$p, e', r, r', r'', r''', o, o', t, t, m, m' \leftarrow\$ (\mathbb{Z}_{q_1})^{12}$

$\mathcal{V}_1 \leftarrow (p, e', r, r', r'', r''', o, o', t, t, m, m')$

$E \leftarrow h^p \tilde{h}^o$

$D \leftarrow g^r \tilde{h}^{o'}$

$A \leftarrow \sigma \tilde{h}^p$

$G \leftarrow g_i \tilde{h}^r$

$W \leftarrow \mathsf{w} \hat{h}^{r'}$

$S \leftarrow \sigma_i \hat{h}^{r''}$

$U \leftarrow u_i \hat{h}^{r'''}$

$m \leftarrow p \cdot c \mod q_1$

$m' \leftarrow r \cdot r'' \mod q_1$

$t \leftarrow o \cdot c \mod q_1$

$t' \leftarrow o' \cdot r'' \mod q_1$

$\mathsf{C}_{\mathsf{rev}} \leftarrow (E, D, A, G, W, S, U, m, m', t, t')$

$\tilde{e}, \tilde{o}, \tilde{o}', \tilde{c}, \tilde{m}, \tilde{m}', \tilde{t}, \tilde{t}', \tilde{\mathsf{m}}_2, \tilde{s}, \tilde{r}, \tilde{r}', \tilde{r}'', \tilde{r}''' \leftarrow\$ (\mathbb{Z}_{q_1})^{14}$

$\mathcal{V}_2 \leftarrow (\tilde{e}, \tilde{o}, \tilde{o}', \tilde{c}, \tilde{m}, \tilde{m}', \tilde{t}, \tilde{t}', \tilde{\mathsf{m}}_2, \tilde{s}, \tilde{r}, \tilde{r}', \tilde{r}'', \tilde{r}''')$

$\bar{T}_1 \leftarrow h^{\tilde{e}} \tilde{h}^{\tilde{o}}$

$\bar{T}_2 \leftarrow E^{\tilde{c}} h^{-\tilde{m}} \tilde{h}^{-\tilde{t}}$

$\bar{T}_3 \leftarrow e(A, \hat{h})^{\tilde{c}} e(\tilde{h}, \hat{h})^{\tilde{r}} e(\tilde{h}, y)^{-\tilde{e}} e(\tilde{h}, \hat{h})^{-\tilde{m}} e(h_1, \hat{h})^{-\tilde{\mathsf{m}}_2} e(h_2, \hat{h})^{-\tilde{s}}$

$\bar{T}_4 \leftarrow e(\tilde{h}, \mathsf{Acc})^{\tilde{r}} e(1/g, \hat{h})^{\tilde{r}'}$

$\bar{T}_5 \leftarrow g^{\tilde{r}} \tilde{h}^{\tilde{o}'}$

$\bar{T}_6 \leftarrow D^{\tilde{r}''} g^{-\tilde{m}'} \tilde{h}^{-\tilde{t}'}$

$\bar{T}_7 \leftarrow e(pk \cdot G, \hat{h})^{\tilde{r}''} e(\tilde{h}, \hat{h})^{-\tilde{m}'} e(\tilde{h}, S)^{\tilde{r}}$

$\bar{T}_8 \leftarrow e(\tilde{h}, u)^{\tilde{r}} e(1/g, \hat{h})^{\tilde{r}'''}$

$\mathsf{T}_{\mathsf{rev}} \leftarrow (\bar{T}_1, \bar{T}_2, \bar{T}_3, \bar{T}_4, \bar{T}_5, \bar{T}_6, \bar{T}_7, \bar{T}_8)$

$\mathsf{M}_{\mathsf{rev}} \leftarrow \{\mathcal{V}_1, \mathcal{V}_2\}$

return $(\mathsf{C}_{\mathsf{rev}}, \mathsf{T}_{\mathsf{rev}}, \mathsf{M}_{\mathsf{rev}})$

Fig. 11: The NIZK proof system $(\mathsf{ZK}_{\mathsf{rev}}.\mathsf{Prove1}, \mathsf{ZK}_{\mathsf{rev}}.\mathsf{Prove2}, \mathsf{ZK}_{\mathsf{rev}}.\mathsf{Verify})$.

$\mathsf{ZK}_{\mathsf{rev}}.\mathsf{Prove2}(\mathsf{r}_{\mathtt{cred}}, \mathsf{M}_{\mathsf{rev}}, c_H)$

parse $\mathsf{r}_{\mathtt{cred}}$ as $(\sigma, c, v_r'', \mathsf{w}_\sigma, g_i, i, \mathsf{m}_2)$

$\hat{e} \leftarrow \tilde{e} - c_H \cdot p \mod q_1$

$\hat{o} \leftarrow \tilde{o} - c_H \cdot o \mod q_1$

$\hat{c} \leftarrow \tilde{c} - c_H \cdot c \mod q_1$

$\hat{o}' \leftarrow \tilde{o}' - c_H \cdot o' \mod q_1$

$\hat{m} \leftarrow \tilde{m} - c_H \cdot m \mod q_1$

$\hat{m}' \leftarrow \tilde{m}' - c_H \cdot m' \mod q_1$

$\hat{t} \leftarrow \tilde{t} - c_H \cdot t \mod q_1$

$\hat{t}' \leftarrow \tilde{t}' - c_H \cdot t' \mod q_1$

$\hat{\mathsf{m}}_2 \leftarrow \tilde{\mathsf{m}}_2 - c_H \cdot \mathsf{m}_2 \mod q_1$

$\hat{s} \leftarrow \tilde{s} - c_H \cdot v_r'' \mod q_1$

$\hat{r} \leftarrow \tilde{r} - c_H \cdot r \mod q_1$

$\hat{r}' \leftarrow \tilde{r}' - c_H \cdot r' \mod q_1$

$\hat{r}'' \leftarrow \tilde{r}'' - c_H \cdot r'' \mod q_1$

$\hat{r}''' \leftarrow \tilde{r}''' - c_H \cdot r''' \mod q_1$

$\rho_{\mathsf{rev}} \leftarrow (\hat{e}, \hat{o}, \hat{c}, \hat{o}', \hat{m}, \hat{m}', \hat{t}, \hat{t}', \hat{\mathsf{m}}_2, \hat{s}, \hat{r}, \hat{r}', \hat{r}'', \hat{r}''')$

return ρ_{rev}

$\mathsf{ZK}_{\mathsf{rev}}.\mathsf{Verify}(\mathsf{rpk}, \mathsf{C}_{\mathsf{rev}}, \rho_{\mathsf{rev}}, c_H, \mathsf{Acc})$

$\hat{T}_1 \leftarrow E^{c_H} h^{\hat{e}} \tilde{h}^{\hat{o}}$

$\hat{T}_2 \leftarrow E^{\hat{c}} h^{-\hat{m}} \tilde{h}^{-\hat{t}}$

$\hat{T}_3 \leftarrow (\frac{e(h_0 G, \hat{h})}{e(A, y)})^{c_H} e(A, \hat{h})^{\hat{c}} e(\tilde{h}, \hat{h})^{\hat{r}} e(\tilde{h}, y)^{-\hat{e}} e(\tilde{h}, \hat{h})^{-\hat{m}}$
$e(h_1, \hat{h})^{-\hat{\mathsf{m}}_2} e(h_2, \hat{h})^{-\hat{s}}$

$\hat{T}_4 \leftarrow (\frac{e(G, \mathsf{Acc})}{e(g, W) \cdot z})^{c_H} e(\tilde{h}, \mathsf{Acc})^{\hat{r}} e(1/g, \hat{h})^{\hat{r}'}$

$\hat{T}_5 \leftarrow D^{c_H} g^{\hat{r}} \tilde{h}^{\hat{o}'}$

$\hat{T}_6 \leftarrow D^{\hat{r}''} g^{-\hat{m}'} \tilde{h}^{-\hat{t}'}$

$\hat{T}_7 \leftarrow (\frac{e(pkG, S)}{e(g, g')})^{c_H} e(pk \cdot G, \hat{h})^{\hat{r}''} e(\tilde{h}, \hat{h})^{-\hat{m}'} e(\tilde{h}, S)^{\hat{r}}$

$\hat{T}_8 \leftarrow (\frac{e(G, u)}{e(g, U)})^{c_H} e(\tilde{h}, u)^{\hat{r}} e(1/g, \hat{h})^{\hat{r}'''}$

$\hat{T}_{\mathsf{rev}} \leftarrow (\hat{T}_1, \hat{T}_2, \hat{T}_3, \hat{T}_4, \hat{T}_5, \hat{T}_6, \hat{T}_7, \hat{T}_8)$

return $\hat{T}_{\mathsf{rev}}$

Fig. 12: More alogorithms for the NIZK proof system $(\mathsf{ZK}_{\mathsf{rev}}.\mathsf{Prove1}, \mathsf{ZK}_{\mathsf{rev}}.\mathsf{Prove2}, \mathsf{ZK}_{\mathsf{rev}}.\mathsf{Verify})$.

4 The HLAC Protocol

In this section, we describe the Hyperledger AnonCreds (HLAC) protocol, as defined in the AnonCreds specification [26], describing the setup, credential issuance and proof presentation phases in turn.

4.1 Setup

The ledger for the HLAC protocol is normally instantiated as an instance of the Hyperledger Indy blockchain [29], a permissioned, append-only ledger that anyone with permission can write to. We capture this in our modelling as follows. We define the ledger as an append-only list and facilitate the initialisation of an empty ledger via algorithm LSetup, as described in Section 2.1. We assume that any entity can setup as an issuer by running algorithm LPost to write AnonCreds objects to the ledger. Additionally, any entity can retrieve the contents of the ledger by running algorithm LRetrieve. We now describe issuer and holder setup, where issuer setup consists of both credential setup and revocation setup. We define formal algorithms for these steps in Figures 13 and 14.

Holder Setup. Holders run algorithm HSetup to generate an empty credential list and a master secret that is used (during credential issuance) to bind credentials to the holder. More specifically, when constructing identity proofs, the holder can prove, in zero-knowledge, that they know the master secret for each credential used in the proof, and that all credentials used in an identity proof are linked to the same master secret. Note that the HLAC protocol does not prevent users from creating multiple master secrets. However, a holder cannot later prove that credentials issued with respect to different master secrets are linked to the same holder.

Credential Setup. Issuers run algorithm Schema to construct a credential schema, which defines the set of attributes that will be used as the basis of a credential definition. On input a list of attributes $\{\mathsf{a}_2, \ldots, \mathsf{a}_L\}$, algorithm Schema assigns the set $\{\mathsf{a}_1, \ldots, \mathsf{a}_L\}$ to S, where a_1 is reserved as a blinded version of the holder's master secret. Algorithm Schema then runs LPost to write the schema to the ledger and generate a Schema ID $\mathtt{ID}_\mathsf{S}$, and outputs $(\mathtt{ID}_\mathsf{S}, \mathsf{S})$.

After a schema is posted to the ledger, an issuer can proceed to generate a credential definition by running algorithm DSetup, which proceeds in the following way. Firstly, algorithm DSetup runs CL.KGen to generate a CL signature key pair $(\mathsf{ppk}, \mathsf{psk})$, which we will refer to as the primary key pair, and associated metadata pmd. The primary public key $\mathsf{ppk} = (n, S, Z, \{R_1, \ldots, R_L\})$, the primary secret key $\mathsf{psk} = (p, q)$ and the metadata $\mathsf{pmd} = (x_z, \{x_i\}_{i \in L})$, i.e., a set of random values from the interval $[2, p'q' - 1]$ used to construct the primary public key values Z and $\{R_1, \ldots, R_L\}$. Secondly, if the issuer sets revocation flag $b = 1$, algorithm DSetup runs algorithm RevKGen to generate a cryptographic accumulator key pair $(\mathsf{rpk}, \mathsf{rsk})$, which we will call the revocation key pair. In particular, $\mathsf{rpk} = (\mathsf{pp}, g, g', h, h_0, h_1, h_2, \tilde{h}, \hat{h}, u, pk, y)$. Thirdly, algorithm DSetup

runs algorithm $\mathsf{ZK}_{\mathsf{ppk}}.\mathsf{Prove}$ to generate a NIZK proof of correct key construction $\rho_{\mathtt{pDef}}$, proving the following relation:

$$\mathcal{R} = \big\{(x_z, \{x_i\}_{i \in L}) \ : \ Z = S^{x_z}, \{R_i = S^{x_i}\}_{i \in L}\big\}.$$

Algorithm DSetup then constructs a public credential definition $\mathtt{pDef} = (\mathtt{ID}_{\mathtt{S}}, \mathsf{ppk}, \mathsf{rpk})$ that includes the schema ID, ensuring a link to the schema and, specifically, the attributes included in the credential. A corresponding private credential definition $\mathtt{sDef} = (\mathsf{psk}, \mathsf{rsk}, \rho_{\mathtt{pDef}})$. Finally, the public credential definition is posted to the ledger and a credential definition ID $\mathtt{ID}_{\mathtt{D}}$ is created by running algorithm LPost. Algorithm DSetup returns the tuple $(\mathtt{ID}_{\mathtt{D}}, \mathtt{pDef}, \mathtt{sDef})$.

$\mathsf{Schema}(\{\mathtt{a}_2, \ldots, \mathtt{a}_L\})$

1 : $\mathtt{S} \leftarrow \{\mathtt{a}_1, \ldots, \mathtt{a}_L\}$
2 : $\mathtt{ID}_{\mathtt{S}} \leftarrow \mathsf{LPost}(\mathtt{S})$
3 : **return** $(\mathtt{ID}_{\mathtt{S}}, \mathtt{S})$

$\mathsf{HSetup}()$

1 : $\mathtt{ms} \leftarrow\!\$\, \{0,1\}^{256}$
2 : $\mathtt{W} \leftarrow \emptyset$
3 : **return** $(\mathtt{ms}, \mathtt{W})$

$\mathsf{DSetup}(\mathtt{ID}_{\mathtt{S}}, \mathtt{S}, \mathtt{b})$

1 : $(\mathsf{ppk}, \mathsf{psk}, \mathsf{pmd}) \leftarrow \mathsf{CL.KGen}(|\mathtt{S}|)$
2 : **if** $b = 0$: $(\mathsf{rpk}, \mathsf{rsk}) \leftarrow (\bot, \bot)$
3 : **if** $b = 1$: $(\mathsf{rpk}, \mathsf{rsk}) \leftarrow\!\$\, \mathsf{RevKGen}()$
4 : $\rho_{\mathtt{pDef}} \leftarrow\!\$\, \mathsf{ZK}_{\mathsf{ppk}}.\mathsf{Prove}((\mathsf{ppk}, \mathtt{S}), (\mathsf{psk}, \mathsf{pmd}))$
5 : $\mathtt{pDef} \leftarrow (\mathtt{ID}_{\mathtt{S}}, \mathsf{ppk}, \mathsf{rpk})$
6 : $\mathtt{sDef} \leftarrow (\mathsf{psk}, \mathsf{rsk}, \rho_{\mathtt{pDef}})$
7 : $\mathtt{ID}_{\mathtt{D}} \leftarrow \mathsf{LPost}(\mathtt{pDef})$
8 : **return** $(\mathtt{ID}_{\mathtt{D}}, \mathtt{pDef}, \mathtt{sDef})$

Fig. 13: Issuer and holder setup algorithms.

Revocation Setup. If revocation is enabled for a credential, i.e., algorithm DSetup generated a revocation key pair, the issuer must run algorithm RSetup to generate a public and private revocation registry and an accumulator tuple. On input a credential definition ID $\mathtt{ID}_{\mathtt{D}}$ and a maximum number of credentials that can be issued $\mathtt{max}$, algorithm RSetup runs algorithm $\mathsf{AccKGen}$ to obtain the tuple $(\mathsf{apk}, \mathtt{sReg}, \mathsf{T}, \mathsf{Acc}, \mathsf{RL})$, where the elements of the tuple are defined as follows. RL is an initial revocation list of length $\mathtt{max}$ where a 1 at position $\mathtt{i}$ indicates that the credential indexed at position $\mathtt{i}$ is revoked, and 0 indicates that the credential is active. Initially, all credentials are revoked and $\mathsf{RL} = \mathbf{1}$. An accumulator Acc is initialised as 0, and a tails file T is generated that contains a static value for each of the $\mathtt{max}$ credentials that can be issued. Additionally, algorithm $\mathsf{AccKGen}$ generates a public/private accumulator key pair $(\mathsf{apk}, \mathtt{sReg})$ where apk is used to generate the tails file and $\mathtt{sReg}$ is used to add and remove credentials from the revoked credentials list. Algorithm RSetup then defines the public revocation registry $\mathtt{pReg}$ as the tuple $(\mathtt{ID}_{\mathtt{D}}, \mathtt{max}, \mathsf{apk}, \mathsf{T}, \mathsf{H}_{\mathsf{T}})$ where H_{T} is a hash of the tails file. Usually, the public revocation registry includes a URL location for the tails file, rather than the tails file itself. For simplicity in our syntax, we include the tails file in the public revocation registry. The public revocation registry is posted to the ledger, creating a revocation registry ID $\mathtt{ID}_{\mathtt{R}}$, and an accumulator

tuple $\mathtt{A} = (\mathtt{ID_R}, \mathsf{Acc}, \mathsf{RL})$ is also posted, creating accumulator ID $\mathtt{ID_A}$. Algorithm RSetup outputs the tuple $(\mathtt{ID_R}, \mathtt{pReg}, \mathtt{sReg}, \mathtt{ID_A}, \mathtt{A})$.

When issuing and revoking credentials, the issuer must update the cryptographic accumulator and post the updated accumulator to the ledger. Doing so ensures that, when verifying identity proofs, a verifier can check that the credentials used in the identity proof are valid with respect to the most recent and up-to-date accumulator, or, indeed, any accumulator of the verifier's choosing. To facilitate this, issuers run algorithm RUpdate, which runs algorithm AccUpd to update the revocation list and accumulator. In particular, AccUpd updates the revocation list to set issued credentials to status 'active' and revoked credentials to status 'revoked'. In other words, if issued credentials list $\mathtt{C_I}[\mathtt{i}] = 1$, indicating that the credential indexed at position i has been issued since the last accumulator update, then revocation list $\mathsf{RL}[\mathtt{i}] \leftarrow 0$. Correspondingly, if revoked list $\mathtt{C_R}[\mathtt{i}] = 1$, then $\mathsf{RL}[\mathtt{i}] \leftarrow 1$. A new accumulator tuple that contains the updated cryptographic accumulator and revocation list is posted to the ledger. The algorithm returns the ID of the updated accumulator $\mathtt{ID_A}$ and new accumulator tuple $\mathtt{A}'$.

RSetup($\mathtt{ID_D}, \mathtt{max}$)

1 : $(\mathsf{apk}, \mathtt{sReg}, \mathsf{T}, \mathsf{Acc}, \mathsf{RL}) \leftarrow \mathsf{AccKGen}(\mathtt{max})$
2 : $\mathsf{H_T} \leftarrow \mathcal{H}(\mathsf{T})$
3 : $\mathtt{pReg} \leftarrow (\mathtt{ID_D}, \mathtt{max}, \mathsf{apk}, \mathsf{T}, \mathsf{H_T})$
4 : $\mathtt{ID_R} \leftarrow \mathsf{LPost}(\mathtt{pReg})$
5 : $\mathtt{A} \leftarrow (\mathtt{ID_R}, \mathsf{Acc}, \mathsf{RL})$
6 : $\mathtt{ID_A} \leftarrow \mathsf{LPost}(\mathtt{A})$
7 : **return** $(\mathtt{ID_R}, \mathtt{pReg}, \mathtt{sReg}, \mathtt{ID_A}, \mathtt{A})$

RUpdate($\mathtt{A}, \mathtt{pReg}, \mathtt{sReg}, \mathtt{C_I}, \mathtt{C_R}$)

1 : **parse** A as $(\mathtt{ID_R}, \mathsf{Acc}, \mathsf{RL})$
2 : $(\mathsf{Acc}', \mathsf{RL}) \leftarrow \mathsf{AccUpd}(\mathsf{Acc}, \mathtt{max}, \mathtt{C_I}, \mathtt{C_R}, \mathtt{sReg})$
3 : $\mathtt{A}' \leftarrow (\mathtt{ID_R}, \mathsf{Acc}', \mathsf{RL})$
4 : $\mathtt{ID_A} \leftarrow \mathsf{LPost}(\mathtt{A}')$
5 : **return** $(\mathtt{ID_A}, \mathtt{A}')$

Fig. 14: Revocation registry algorithms.

4.2 Credential Issuance

The credential issuance phase is a process between an issuer and a holder, in which the issuer generates a credential for the holder. The issuer initiates the

credential issuance phase by generating a credential offer, which states the credential the issuer is willing to offer and includes the zero-knowledge proof of correct key construction output by algorithm DSetup. The holder responds with a credential request, which comprises a Pedersen commitment to the holder's master secret and a zero-knowledge proof of correct blinding. The issuer then issues the credential and the holder stores the credential in their credential list. We now describe each step in the credential issuance protocol in detail, and formally define the algorithms in Figures 15 and 16.

$\mathsf{Offer}(\mathtt{ID_S}, \mathtt{ID_D}, \mathtt{sDef})$

1 : **parse** $\mathtt{sDef}$ as $(\mathsf{psk}, \mathsf{rsk}, \rho_{\mathtt{pDef}})$
2 : $n_0 \leftarrow\$ \{0,1\}^{80}$
3 : **return** $\mathtt{Offer} \leftarrow (\mathtt{ID_S}, \mathtt{ID_D}, n_0, \rho_{\mathtt{pDef}})$

$\mathsf{Request}(\mathtt{Offer}, \mathtt{ms}, \{v_2, \ldots, v_L\})$

1 : **parse** $\mathtt{Offer}$ as $(\mathtt{ID_S}, \mathtt{ID_D}, n_0, \rho_{\mathtt{pDef}})$
2 : **parse** pp as $(G_1, G_2, G_T, e, q_1, g, g')$
3 : $\mathtt{pDef} \leftarrow \mathsf{LRetrieve}(\mathtt{ID_D})$
4 : **parse** $\mathtt{pDef}$ as $(\mathtt{ID_S}, \mathsf{ppk}, \mathsf{rpk})$
5 : **parse** ppk as $(n, S, Z, \{R_1, \ldots, R_L\})$
6 : $\mathtt{S} \leftarrow \mathsf{LRetrieve}(\mathtt{ID_S})$
7 : **if** $\mathsf{ZK}_{\mathsf{ppk}}.\mathsf{Verify}((\mathsf{ppk}, \mathtt{S}), \rho_{\mathtt{pDef}}) \neq 1$ **return** $\perp$
8 : $n_1 \leftarrow\$ \{0,1\}^{80}$
9 : $\mathsf{ent} \leftarrow \{0,1\}^{256}$
10 : $\mathsf{U} \leftarrow \mathsf{Commit}_{(S,R_1)}(\mathtt{ms}; v')$
11 : $\rho_{\mathsf{U}} \leftarrow\$ \mathsf{ZK}_{\mathtt{ms}}.\mathsf{Prove}((\mathsf{ppk}, \mathsf{U}, n_0), (v', \mathtt{ms}))$
12 : **if** $\mathsf{rpk} = \perp$: $u_r \leftarrow \perp$
13 : **if** $\mathsf{rpk} \neq \perp$
14 : **parse** rpk as $(\mathsf{pp}, g, g', h, h_0, h_1, h_2, \tilde{h}, \hat{h}, u, pk, y)$
15 : $s'_r \leftarrow \mathcal{QR}_{q_1}$
16 : $u_r \leftarrow h_2^{s'_r}$
17 : $\mathsf{BlindMS} \leftarrow (\mathsf{U}, u_r)$
18 : $\mathtt{pReq} \leftarrow (\mathtt{ID_D}, \{v_2, \ldots, v_L\}, \mathsf{BlindMS}, \rho_{\mathsf{U}}, \mathsf{ent}, n_1)$
19 : $\mathtt{sReq} \leftarrow (v', s'_r)$
20 : **return** $(\mathtt{pReq}, \mathtt{sReq})$

Fig. 15: Algorithms in the Credential Issuance phase.

$\mathsf{Issue}(\mathtt{pReq}, \mathtt{Offer}, \mathtt{pDef}, \mathtt{sDef}, \mathtt{pReg}, \mathtt{sReg}, \mathtt{ID_A}, \mathtt{A}, \mathtt{S}, i)$

1 : **parse** $\mathtt{pReq}$ as $(\mathtt{ID_D}, \{v_2, \ldots, v_L\}, \mathsf{BlindMS}, \rho_{\mathsf{U}}, \mathsf{ent}, n_1)$
2 : **parse** $\mathtt{Offer}$ as $(\mathtt{ID_S}, \mathtt{ID_D}, n_0, \rho_{\mathtt{pDef}})$
3 : **parse** $\mathsf{BlindMS}$ as (U, u_r)
4 : **parse** $\mathtt{pDef} = (\mathtt{ID_S}, \mathsf{ppk}, \mathsf{rpk})$
5 : **parse** $\mathtt{sDef}$ as $(\mathsf{psk}, \mathsf{rsk})$
6 : **parse** $\mathtt{pReg}$ as $(\mathtt{ID_D}, \mathtt{max}, \mathsf{apk}, \mathsf{T}, \mathsf{H_T})$
7 : **parse** $\mathtt{A}$ as $(\mathtt{ID_R}, \mathsf{Acc}, \mathsf{RL})$
8 : **parse** $\mathtt{S}$ as $\{\mathsf{a}_1, \ldots, \mathsf{a}_L\}$
9 : **if** $\mathsf{ZK}_{\mathtt{ms}}.\mathsf{Verify}((\mathsf{ppk}, \mathsf{U}, n_0), \rho_{\mathsf{U}}) \neq 1$ **return** $\perp$
10 : $v_1 \leftarrow \mathsf{U}$
11 : $\mathsf{m}_2 \leftarrow \mathcal{H}(i \| \mathsf{ent})$
12 : $(\sigma, Q) \leftarrow \mathsf{CL.Sign}(\mathsf{ppk}, \mathsf{psk}, \{v_1, \ldots, v_L\})$
13 : $\sigma_p \leftarrow (\mathsf{m}_2, \sigma)$
14 : $\rho_\sigma \leftarrow\!\!\$\ \mathsf{ZK}_\sigma.\mathsf{Prove}((\sigma, n_1), (Q, \mathsf{psk}))$
15 : $(\mathsf{r}_{\mathtt{cred}}, \mathsf{w}) \leftarrow\!\!\$\ \mathsf{AccAdd}(\mathsf{rpk}, \mathsf{rsk}, \mathtt{sReg}, \mathtt{max}, i, \mathsf{RL}, \mathsf{m}_2, u_r)$
16 : $(\mathtt{ID_A}, \mathtt{A}') \leftarrow \mathsf{RUpdate}(\mathtt{A}, \mathtt{pReg}, \mathtt{sReg}, i, \perp)$
17 : $\mathtt{cred} \leftarrow (\mathtt{ID_S}, \mathtt{ID_D}, \mathtt{ID_R}, \{(\mathsf{a}_i, v_i)\}_{i \in [L]}, \sigma_p, \rho_\sigma, \mathsf{r}_{\mathtt{cred}}, \mathtt{ID_A}, \mathsf{w})$
18 : **return** $\mathtt{cred}$

$\mathsf{Credential.Store}(\mathtt{cred}, \mathtt{pReq}, \mathtt{sReq}, \mathtt{W})$

1 : **parse** $\mathtt{cred}$ as $(\mathtt{ID_S}, \mathtt{ID_D}, \mathtt{ID_R}, \{(\mathsf{a}_i, v_i)\}_{i \in [L]}, \sigma_p, \rho_\sigma, \mathsf{r}_{\mathtt{cred}}, \mathtt{ID_A}, \mathsf{w})$
2 : **parse** $\mathtt{pReq}$ as $(\mathtt{ID_D}, \{v_2, \ldots, v_L\}, \mathsf{BlindMS}, \rho_{\mathsf{U}}, \mathsf{ent}, n_1)$
3 : $\mathtt{pDef} \leftarrow \mathsf{LRetrieve}(\mathtt{ID_D})$
4 : **parse** $\mathtt{pDef} = (\mathtt{ID_S}, \mathsf{ppk}, \mathsf{rpk})$
5 : **parse** $\mathtt{sReq}$ as (v', s'_r)
6 : **parse** σ_p as (m_2, σ)
7 : **if** $\mathsf{ZK}_\sigma.\mathsf{Verify}((\sigma, n_1), (\mathsf{ppk}, \mathtt{sReq}, \{v_i\}_{i \in [L]}), \rho_\sigma) \neq 1$ **return** $\perp$
8 : $\mathtt{W} \leftarrow \mathtt{W} \cup \mathtt{cred}$

Fig. 16: Algorithms in the Credential Issuance phase.

Credential Offer. The issuer runs algorithm Offer, which takes as input a credential definition and schema ID ($\mathtt{ID_S}$ and $\mathtt{ID_D}$) and the private credential definition $\mathtt{sDef}$. Algorithm Offer generates a nonce n_0 that is used to generate a zero-knowledge proof for the credential request, tying the credential request to the offer and preventing replay attacks. The algorithm returns an offer $\mathtt{Offer} = (\mathtt{ID_S}, \mathtt{ID_D}, n_0, \rho_{\mathtt{pDef}})$ where $\rho_{\mathtt{pDef}}$ is the proof of correct key construction in $\mathtt{sDef}$.

Credential Request. The holder responds to a credential offer by creating a credential request, the main purpose of which is to create a blinded version of the

master secret that will be set as the first, reserved, attribute of the credential v_1. Formally, the issuer runs algorithm Request, which takes as input a credential offer $\texttt{Offer}$, a master secret $\texttt{ms}$ and a set of attribute values $\{v_2, \ldots, v_L\}$. Algorithm Request first runs algorithm $\mathsf{ZK}_{\mathsf{ppk}}$.Verify to verify the proof of correct key construction $\rho_{\texttt{pDef}}$ included in $\texttt{Offer}$. If the proof verifies, a credential request is generated. Firstly, a Pedersen commitment to the master secret $\texttt{ms}$ is generated by running algorithm Commit, which outputs a blinded master secret U. Algorithm Request also generates a NIZK proof of correct blinding by running algorithm $\mathsf{ZK}_{\texttt{ms}}$.Prove, which proves the relation:

$$\mathcal{R} = \left\{(\texttt{ms}, v') \ : \ \mathsf{U} = S^{v'} \cdot R_1^{\texttt{ms}}\right\}$$

where v' is the randomness used to generate the commitment, and elements S and R_1 are drawn from the primary public key ppk. Then, if revocation is enabled for the credential, a blinded revocation secret u_r is computed as $h_2^{s'_r}$ where s'_r is chosen uniformly at random from the set of quadratic residues mod q_1, denoted $\mathcal{QR}_{q_1}$, and h_2 is an element in the revocation public key rpk.

Algorithm Request outputs a public/private request pair $(\texttt{pReq}, \texttt{sReq})$ that is constructed in the following way. The public credential request, transmitted to the issuer is $\texttt{pReq} = (\texttt{ID}_\texttt{D}, \{v_2, \ldots, v_L\}, \mathsf{BlindMS}, \rho_\mathsf{U}, ent, n_1)$, where $\texttt{ID}_\texttt{D}$ is obtained from the credential offer and BlindMS contains the blinded master secret and the blinded revocation secret. That is, $\mathsf{BlindMS} = (\mathsf{U}, u_r)$. Moreover, n_1 and *ent* are random strings used during credential issuance, with n_1 used to tie the request to the issued credential, and $\texttt{ent}$ used to form the credential signature. The private credential request contains the randomness used to generate the blinded master secret and blinded revocation secret (i.e., $\texttt{sReq} = (v', s'_r)$), and is stored by the holder and used to store the credential issued in the next stage.

Issue Credential. The issuer can now issue credentials. To run algorithm Issue, the issuer must collect the credential offer $\texttt{Offer}$, public credential request $\texttt{pReq}$, the public and private credential definition $(\texttt{pDef}, \texttt{sDef})$, the public and private revocation registry $(\texttt{pReg}, \texttt{sReg})$, the accumulator $\texttt{A}$, corresponding accumulator ID $\texttt{ID}_\texttt{A}$, and the schema $\texttt{S}$. The issuer must also select an index $\texttt{i}$ that determines the revocation registry index of the credential. Specifically, the issued credential will be associated with the $\texttt{i}$'th static element of the tails fails T constructed during revocation setup. Taking these values as input, algorithm Issue first runs algorithm $\mathsf{ZK}_{\texttt{ms}}$.Verify to verify the NIZK proof of correct blinding in the credential request. If the proof verifies, algorithm Issue computes the credential. Algorithm Issue assigns attribute value v_1 as the blinded master secret U, and computes the credential context m_2 as the hash of index $\texttt{i}$ concatenated with the string *ent* included in the credential request. Then, a CL signature $\sigma = (A, e, v'')$ and signing metadata Q are generated by running algorithm CL.Sign. Algorithm ZK_σ.Prove then generates a NIZK proof of correct signing ρ_σ, proving the following relation:

$$\mathcal{R} = \left\{(e^{-1}) \ : \ A = Q^{e^{-1}}\right\}.$$

The signature for the credential $\sigma_p = (\sigma, \mathsf{m}_2)$. Algorithm Issue then proceeds to update the revocation status of the credential by running algorithm AccAdd to obtain a revocable credential $\mathsf{r}_{\mathtt{cred}}$ and witness w. Then, algorithm RUpdate updates the accumulator, returning an ID for the new accumulator tuple $\mathtt{ID}_\mathtt{A}$. Algorithm Issue returns a credential $\mathtt{cred} = (\mathtt{ID}_\mathtt{S}, \mathtt{ID}_\mathtt{D}, \mathtt{ID}_\mathtt{R}, \{(\mathsf{a}_i, v_i)\}_{i\in[L]}, \sigma_p, \rho_\sigma, \mathsf{r}_{\mathtt{cred}}, \mathtt{ID}_\mathtt{A}, \mathsf{w})$.

Store Credential. The holder can now store the credential in their credential list. Algorithm Store facilitates this by running algorithm ZK_σ.Verify to verify the NIZK proof of correct signing ρ_σ. If the proof verifies, algorithm Store adds the credential to the list W.

4.3 Credential Presentation

The credential presentation phase allows a holder to generate privacy-preserving identity proofs pertaining to their credentials that can be verified by third party verifiers. Specifically, holders generate NIZK proofs on their credentials that reveal only a subset of attributes and/or predicates based on their attribute values. The verifier initiates the credential presentation phase by issuing a presentation request that defines the desired identity proof. The holder responds with a proof presentation message, which contains an identity proof that the verifier can verify. We now provide details of these steps, formalising the algorithms in Figures 17 and 18.

Identity Proof Request. To generate an identity proof request, the verifier must construct sets R and P that are input to algorithm Propose. These sets define the data that the holder must collect in order to generate their identity proof. They detail the attributes and predicates that the identity proof should reveal, restrictions on the credential definitions to be used, and limitations on freshness of revocation status. More specifically, set R consists of tuples of the form $(\mathsf{a}, \mathtt{ID}_\mathtt{D}, \mathtt{ID}_\mathtt{A})$ where a is an attribute that the verifier wants the holder to reveal, $\mathtt{ID}_\mathtt{D}$ is the ID for a credential definition that contains attribute a and should be used to generate the identity proof, and $\mathtt{ID}_\mathtt{A}$ is the ID for the accumulator tuple that should be used to provide the proof of non-revocation. The set P is a set of tuples of the form $(\mathsf{a}, p, z, \mathtt{ID}_\mathtt{D}, \mathtt{ID}_\mathtt{A})$. Here, a, $\mathtt{ID}_\mathtt{D}$ and $\mathtt{ID}_\mathtt{A}$ are defined as for set R. Additionally, $p \in \{<, \leq, >, \geq\}$ is a predicate that the holder must prove for attribute a and z is an integer for which the predicate must hold. For example, if $p =<$, attribute value v corresponding to attribute a must be less than z. On input sets R and P, algorithm Propose generates a random nonce n_2 that is used by the holder to generate the identity proof, and outputs the presentation proposal $\mathtt{PReq} = (\mathtt{R}, \mathtt{P}, n_2)$.

In the HLAC protocol, the verifier can request multiple revealed attributes and predicates from a single credential. We capture this in our modelling, requiring that the verifier defines a new tuple in R or P for each attribute and predicate. Moreover, according to the HLAC specification, verifiers can define a number of restrictions on the tuples contained in sets R and P. For example, the

$\mathsf{Propose}(\mathtt{R}, \mathtt{P})$

$n_2 \leftarrow\$ \{0,1\}^{80}$

return $\mathtt{PReq} = (\mathtt{R}, \mathtt{P}, n_2)$

$\mathsf{Present}(\mathtt{PReq}, \mathsf{C_P}, \mathtt{ms})$

parse $\mathtt{PReq}$ as $(\mathtt{R}, \mathtt{P}, n_2)$

parse $\mathsf{C_P}$ as $(\{(\mathtt{cred}_i, A_{r,i}, A_{\bar{r},i}, V_{r,i}, \mathtt{P}'_i, \mathtt{ID}'_{\mathtt{A}_1}, \mathtt{A}'_i, \mathtt{A}_i, \mathtt{pDef}_i, \mathtt{pReg}_i)\}_{i\in[|\mathsf{C_P}|]})$

for $i \in 1, \ldots, |\mathsf{C_P}|$

  for $j \in A_{\bar{r},i}$: $\hat{v}_{i,j} \leftarrow\$ \{0,1\}^{592}$

  parse $\mathtt{cred}_i$ as $(\mathtt{ID}_{\mathtt{S}_i}, \mathtt{ID}_{\mathtt{D}_i}, \mathtt{ID}_{\mathtt{R}_i}, \{(\mathtt{a}_{i,j}, \mathtt{v}_{i,j})\}_{j\in[L]}, \sigma_{p_i}, \rho_{\sigma_i}, \mathsf{r}_{\mathtt{cred}_i}, , \mathtt{ID}_{\mathtt{A}_i}, \mathsf{w}_i)$

  parse σ_{p_i} as $(\mathsf{m}_{2_i}, \sigma_i)$

parse $\mathtt{pDef}_i$ as $(\mathtt{ID}_{\mathtt{S}_i}, \mathsf{ppk}_i, \mathsf{rpk}_i)$ $\wedge$ $\mathtt{sReg}$ as $(\mathtt{ID}_{\mathtt{D}_i}, \mathtt{max}_i, \mathsf{apk}_i, \mathsf{T}_i, \mathsf{H}_{\mathsf{T}_i})$

  parse $\mathtt{P}'_i$ as $(\{(k_{i,j}, p_{i,j}, z_{i,j}, \mathtt{v}_{i,k_{i,j}})\}_{j\in[|\mathtt{P}'_i|]})$

parse $\mathtt{A}'_i$ as $(\mathtt{ID}_{\mathtt{R}_i}, \mathsf{Acc}'_i, \mathsf{RL}'_i)$ $\wedge$ $\mathtt{A}_i$ as $(\mathtt{ID}_{\mathtt{R}_i}, \mathsf{Acc}_i, \mathsf{RL}_i)$

  $(\mathsf{C}_{\mathtt{cred}_i}, \mathsf{T}_{\mathtt{cred}_i}) \leftarrow\$ \mathsf{ZK}_{\mathtt{cred}}.\mathsf{Prove1}((A_{\bar{r},i}, \mathsf{ppk}_i), (\sigma_i, \{\hat{v}_{i,j}\}_{j\in A_{\bar{r},i}}))$

  for $j \in 1, \ldots, |\mathtt{P}'_i|$

    $(\mathsf{C}_{\mathsf{pred}_{i,j}}, \mathsf{T}_{\mathsf{pred}_{i,j}}, \mathsf{M}_{\mathsf{pred}_{i,j}}) \leftarrow\$ \mathsf{ZK}_{\mathsf{pred}}.\mathsf{Prove1}((\mathsf{ppk}_i, p_{i,j}, z_{i,j}), (v_{i,k_{i,j}}, \hat{v}_{i,k_{i,j}}))$

  $\mathsf{w}'_i \leftarrow \mathsf{Acc.WUpd}(\mathsf{w}_i, \mathsf{RL}_i, \mathsf{RL}'_i, \mathsf{T}_i)$

  $(\mathsf{C}_{\mathsf{rev}_i}, \mathsf{T}_{\mathsf{rev}_i}, \mathsf{M}_{\mathsf{rev}_i}) \leftarrow\$ \mathsf{ZK}_{\mathsf{rev}}.\mathsf{Prove1}(\mathsf{r}_{\mathtt{cred}_i}, \mathsf{pp}, \mathsf{rpk}_i, \mathsf{Acc}'_i)$

  $\mathsf{C}_i \leftarrow (\mathsf{C}_{\mathtt{cred}_i}, \{\mathsf{C}_{\mathsf{pred}_{i,j}}\}_{j\in[|\mathtt{P}'_i|]}, \mathsf{C}_{\mathsf{rev}_i})$; $\mathsf{T}_i \leftarrow (\mathsf{T}_{\mathtt{cred}_i}, \{\mathsf{T}_{\mathsf{pred}_{i,j}}\}_{j\in[|\mathtt{P}'_i|]}, \mathsf{T}_{\mathsf{rev}_i})$

  $\mathtt{P}''_i \leftarrow (\{(k_{i,j}, p_{i,j}, z_{i,j}, \tilde{v}_{i,k_j})\}_{j\in[|\mathtt{P}'_i|]})$

$\mathsf{C} \leftarrow (\mathsf{C}_1, \ldots, \mathsf{C}_{|\mathsf{C_P}|})$; $\mathsf{T} \leftarrow (\mathsf{T}_1, \ldots, \mathsf{T}_{|\mathsf{C_P}|})$; $c_H \leftarrow \mathcal{H}(\mathcal{T}\|\mathcal{C}\|n_2)$

for $i \in 1, \ldots, |\mathsf{C_P}|$

  $\rho_{\mathtt{cred}_i} \leftarrow \mathsf{ZK}_{\mathtt{cred}}.\mathsf{Prove2}((\mathsf{C}_{\mathtt{cred}_i}, A_{\bar{r},i}, c_H), (\mathsf{T}_{\mathtt{cred}_i}, \{\hat{v}_{i,j}, v_{i,j}\}_{j\in A_{\bar{r},i}}))$

  for $j \in 1, \ldots, |\mathtt{P}'_i|$: $\rho_{\mathsf{pred}_{i,j}} \leftarrow\$ \mathsf{ZK}_{\mathsf{pred}}.\mathsf{Prove2}(c_H, \mathsf{M}_{\mathsf{pred}_{i,j}})$

  $\rho_{\mathsf{rev}_i} \leftarrow\$ \mathsf{ZK}_{\mathsf{rev}}.\mathsf{Prove2}(c_H, \mathsf{M}_{\mathsf{rev}_i}, \mathsf{r}_{\mathtt{cred}_i})$

  $\rho_i \leftarrow (\rho_{\mathtt{cred}_i}, (\{\rho_{\mathsf{pred}_{i,j}}\}_{j\in[|\mathtt{P}'_i|]}), \rho_{\mathsf{rev}_i})$

return $\mathtt{Proof} \leftarrow (\mathsf{C}, c_H, (\{(\rho_i, A_{r,i}, A_{\bar{r},i}, V_{r,i}, \mathtt{ID}_{\mathtt{S}_i}, \mathtt{ID}_{\mathtt{D}_i}, \mathtt{ID}_{\mathtt{R}_i}, \mathtt{ID}'_{\mathtt{A}_i}, \mathtt{P}'_i)\}_{i\in[|\mathsf{C_P}|]}))$

Fig. 17: Algorithms in the Proof Presentation phase.

verifier can request that the credential is linked to a particular schema, or define an interval in which a credential must be shown to be active (i.e., not revoked). For clarity and simplicity in our modelling, we set $\mathtt{ID}_{\mathtt{A}}$ and $\mathtt{ID}_{\mathtt{D}}$ as restriction, noting that they are sufficient to capture other restrictions.

Present Identity Proof. The holder generates an identity proof by running algorithm Present. Ahead of this, the holder must collect credentials and related data to be used in the identity proof from their credential list $\mathtt{W}$. For each credential

$\mathsf{Verify}(\mathtt{Proof}, \mathtt{PReq})$

1 : **parse** $\mathtt{Proof}$ as $(\mathsf{C}, c_H, (\{(\rho_i, A_{r,i}, A_{\bar{r},i}, V_{r,i}, \mathtt{ID}_{\mathtt{S}_i}, \mathtt{ID}_{\mathtt{D}_i}, \mathtt{ID}_{\mathtt{R}_i}, \mathtt{ID}'_{\mathtt{A}_i}, \mathtt{P}'_i)\}_{i\in[|\mathsf{C_P}|]}))$
2 : **parse** C as $(\mathsf{C}_1, \ldots, \mathsf{C}_{|\mathsf{C_P}|})$ $\wedge$ $\mathtt{PReq}$ as $(\mathtt{R}, \mathtt{P})$
3 : **for** $i \in 1, \ldots, |\mathsf{C_P}|$
4 : **parse** ρ_i as $(\rho_{\mathsf{cred}_i}, (\{\rho_{\mathsf{pred}_{i,j}}\}_{j\in[|\mathtt{P}'_i|]}), \rho_{\mathsf{rev}_i})$
5 : $\mathtt{pDef}_i \leftarrow \mathsf{LRetrieve}(\mathtt{ID}_{\mathtt{D}_i})$; **parse** $\mathtt{pDef}_i$ as $(\mathtt{ID}_{\mathtt{S}_i}, \mathsf{ppk}_i, \mathsf{rpk}_i)$
6 : **parse** C_i as $(\mathsf{C}_{\mathsf{cred}_i}, \{\mathsf{C}_{\mathsf{pred}_{i,j}}\}_{j\in[|\mathtt{P}'_i|]}, \mathsf{C}_{\mathsf{rev}_i})$
7 : **parse** $\mathtt{P}''_i$ as $(\{(k_{i,j}, p_{i,j}, z_{i,j}, \tilde{v}_{i,k_j})\}_{j\in[|\mathtt{P}'_i|]})$
8 : $\mathtt{A}_i \leftarrow \mathsf{LRetrieve}(\mathtt{ID}'_{\mathtt{A}_i})$; **parse** $\mathtt{A}_i$ as $(\mathtt{ID}_{\mathtt{R}_i}, \mathsf{Acc}_i, \mathsf{RL}_i)$
9 : $\hat{T}_{\mathsf{cred}_i} \leftarrow\!\$\, \mathsf{ZK}_{\mathtt{cred}}.\mathsf{Verify}((A_{\bar{r},i}, A_{r,i}, c_H, V_{r,i}), \mathsf{ppk}_i, \rho_{\mathsf{cred}_i})$
10 : **for** $j = 1, \ldots, |\mathtt{P}''_i|$
11 : $\hat{T}_{\mathsf{pred}_i} \leftarrow\!\$\, \mathsf{ZK}_{\mathsf{pred}}.\mathsf{Verify}((\mathsf{ppk}_i, p_{i,j}, z_{i,j}, \mathsf{C}_{\mathsf{pred}_{i,j}}), c_H, \rho_{\mathsf{pred}_{i,j}}, \tilde{v}_{i,k_j})$
12 : $\hat{T}_{\mathsf{rev}_i} \leftarrow\!\$\, \mathsf{ZK}_{\mathsf{rev}}.\mathsf{Verify}(\mathsf{r}_{\mathsf{cred}_i}, \mathsf{C}_{\mathsf{rev}_i}, \rho_{\mathsf{rev}_i}, c_H, \mathsf{Acc}_i)$
13 : $\hat{\mathsf{T}}_i \leftarrow (\hat{\mathsf{T}}_{\mathsf{cred}_i}, \{\hat{\mathsf{T}}_{\mathsf{pred}_{i,j}}\}_{j\in[|\mathtt{P}'_i|]}, \hat{\mathsf{T}}_{\mathsf{rev}_i})$
14 : $\hat{\mathsf{T}} \leftarrow (\hat{\mathsf{T}}_1, \ldots, \hat{\mathsf{T}}_{|\mathsf{C_P}|})$
15 : **if** $c_H \neq \mathcal{H}(\hat{\mathsf{T}} \| \mathsf{C} \| n_2)$ **return** 0
16 : **return** 1

Fig. 18: Algorithms in the Proof Presentation phase.

to be used in the identity proof, the holder constructs a tuple of the form

$$(\mathtt{cred}, A_r, A_{\bar{r}}, V_r, \mathtt{P}', \mathtt{ID}'_{\mathtt{A}}, \mathtt{A}', \mathtt{A}, \mathtt{pDef}, \mathtt{pReg}).$$

Here, $\mathtt{cred}$ is a credential output by algorithm Issue. Sets A_r and $A_{\bar{r}}$ are disjoint sets that contain the indices of attributes from $\mathtt{cred}$ such that $|A_r \cup A_{\bar{r}}| = L$. Set A_r contains the indices of attributes that will be revealed in the identity proof, and set $A_{\bar{r}}$ contains the indices of hidden attributes. Set V_r contains the attribute values from credential $\mathtt{cred}$ for the revealed attributes. For each credential, the holder also extracts tuples from set $\mathtt{P}$ that pertains to credential $\mathtt{cred}$ and collects the tuples in a set $\mathtt{P}'$. Additionally, set $\mathtt{P}'$ includes the attribute value and index of the attribute value that will be used to prove the predicate. The holder then collects the public credential definition $\mathtt{pDef}$ and revocation registry $\mathtt{pReg}$, the accumulator tuple used to construct the credential $\mathtt{A}$, and the accumulator tuple $\mathtt{A}'$ corresponding to $\mathtt{ID}'_{\mathtt{A}}$ included in the presentation request. The credential tuples are collected in set $\mathsf{C_P}$ and input to algorithm $\mathsf{Present}$ alongside the holder's master secret $\mathtt{ms}$ and the presentation request $\mathtt{PReq}$. Algorithm $\mathsf{Present}$ then proceeds to construct an identity proof as follows.

Algorithm $\mathsf{Present}$ proves ownership of each credential used in the proof, i.e., knowledge of the master secret included in blinded form in the credential (lines 7 and 15), that each predicate in set $\mathtt{P}$ is true (lines 8 and 16), and demonstrates

the non-revocation status of each credential used in the proof (lines 9, 10 and 17).

Rather than running a NIZK proving algorithm for each of these three steps to generate a commitment, challenge and response individually, algorithm Present generates an *aggregate* proof. In an aggregate proof, a commitment is generated for each proof system, running algorithms $\mathsf{ZK}_{\mathtt{cred}}$.Prove1, $\mathsf{ZK}_{\mathsf{pred}}$.Prove1 and $\mathsf{ZK}_{\mathsf{rev}}$.Prove1. Each of these algorithms outputs a list of C and T elements that are required to compute the hash included in the proof. Additionally, algorithms $\mathsf{ZK}_{\mathsf{pred}}$.Prove1 and $\mathsf{ZK}_{\mathsf{rev}}$.Prove1 return a set of metadata M that is required to generate the identity proof. An aggregate challenge is then computed. As is usual in a NIZK proof system, the challenge is computed as a hash of the protocol transcript. That is, the C and T values output by all commitment algorithms are collected and hashed along with the nonce n_2 generated by the verifier running algorithm Propose(line 13). The response values for the proof are then computed by running algorithms $\mathsf{ZK}_{\mathtt{cred}}$.Prove2, $\mathsf{ZK}_{\mathsf{pred}}$.Prove2 and $\mathsf{ZK}_{\mathsf{rev}}$.Prove2. Outputs of these algorithms are included in the identity proof `Proof` returned by algorithm Present, which also includes the set of C values and other data input to algorithm Present that the verifier requires to verify the identity proof. In particular, the identity proof includes the IDs for all AnonCreds objects used to construct the proof, and includes the sets A_r, $A_{\bar{r}}$ and V_r for each credential used in the proof.

We now present more details on the NIZK proof systems included in the aggregate proof. To demonstrate ownership of each credential, Present generates a random value $\hat{v}$ for each hidden attribute (line 3) and generates the commitment for the following relation:

$$\mathcal{R} = \Big\{(e, \{v_i \colon i \in A_{\bar{r}}\}) \ : \ \frac{Z}{\prod_{i \in A_r} R_i^{v_i}} = A^e \cdot S^{v''} \prod_{i \in A_{\bar{r}}} R_i^{v_i} \mod n \\ \wedge \ \{v_i \in \{0,1\}^{592}\}_{i \in A_{\bar{r}}} \ \wedge \ e - 2^{596} \in \{0,1\}^{458}\Big\}$$

where values used in the proof are drawn from the primary public key ppk, the CL signature σ and the attribute value set $\{v_1, \ldots, v_L\}$.

Then, for each predicate, create a commitment for relation:

$$\mathcal{R} = \Big\{(v, r_\Delta, \{u_1, \ldots, u_4\}, \{r_1, \ldots, r_4\}, a) \ : T_\Delta^a \cdot Z^\Delta = Z^v (S^a)^{r_\Delta} \mod n \\ \wedge \ \{\textbf{for } j \in [4] \ : \ T_j = Z^{u_j} \cdot S^{r_j} \mod n\} \\ \wedge \ T_\Delta = T_1^{u_1} \cdot \cdots \cdot T_4^{u_4} \cdot S^a \mod n\Big\}$$

where v is the attribute value for the predicate, Z and S are drawn from ppk and all other values are generated by running algorithm $\mathsf{ZK}_{\mathsf{pred}}$.Prove1.

The witness used to demonstrate non-revocation must then be updated in order to demonstrate that the credential used in the identity proof is currently active (line 9). A non-revocation commitment, taking as input the updated wit-

ness proves the following relation:

$$\mathcal{R} = \big\{(c, p, o, m, t, \mathsf{m}_2, v_r'', r, o', m', t', r', r'', r''') \; : \\ E = h^p \cdot \tilde{h}^o \;\wedge\; E^c \cdot h^{-m} \cdot \tilde{h}^{-t} = 1 \;\wedge\; D = g^r \cdot \tilde{h}^{o'} \\ \wedge\; \frac{e(h_0 \cdot G, \hat{h})}{e(A, y)} = e(A, \hat{h})^c \cdot e(\tilde{h}, \hat{h})^r \cdot e(\tilde{h}, y)^{-e} \cdot e(\tilde{h}, \hat{h})^{-m} \cdot e(h_1, \hat{h})^{-\mathsf{m}_2} \cdot e(h_2, \hat{h})^{-s} \\ \wedge\; \frac{e(G, \mathsf{Acc})}{e(g, W) \cdot z} = e(\tilde{h}, \mathsf{Acc})^r \cdot e(\frac{1}{g}, \hat{h})^{r'} \\ \wedge\; \frac{e(pk \cdot G, S)}{e(g, g')} = e(pk \cdot G, \hat{h})^{r''} \cdot e(\tilde{h}, \hat{h})^{-m'} \cdot e(\tilde{h}, S)^r \\ \wedge\; \frac{e(G, u)}{e(g, U)} = e(\tilde{h}, u)^r \cdot e(\frac{1}{g}, \hat{h})^{r'''}\big\}$$

where values used in this proof system are taken from the public parameters $\mathsf{pp} = (G_1, G_2, G_T, e, q_1, g, g')$, the revocation public key $\mathsf{rpk} = (\mathsf{pp}, g, g', h, h_0, h_1, h_2, \tilde{h}, \hat{h}, u, pk, y)$, or are generated when running algorithm $\mathsf{ZK}_{\mathsf{rev}}.\mathsf{Prove1}$.

The hash is then computed and algorithm Present runs $\mathsf{ZK}_{\mathtt{cred}}.\mathsf{Prove2}$, $\mathsf{ZK}_{\mathsf{pred}}.\mathsf{Prove2}$ and $\mathsf{ZK}_{\mathsf{rev}}.\mathsf{Prove2}$ to complete the identity proof.

Verify Identity Proof. To verify the identity proof, the verifier runs algorithm Verify which runs algorithms $\mathsf{ZK}_{\mathtt{cred}}.\mathsf{Verify}$, $\mathsf{ZK}_{\mathsf{pred}}.\mathsf{Verify}$ and $\mathsf{ZK}_{\mathsf{rev}}.\mathsf{Verify}$ to recompute the T values for the identity proof. The holder then recomputes the hash included in the identity proof, using the C values included in the proof, the re-computed T values and the nonce n_2 included in the presentation request. If the recomputed hash is equal to the hash included in the identity proof, algorithm Verify returns 1 and the identity proof is said to be valid.

5 Security Analysis

We demonstrate that the HLAC protocol described in Section 4 is correct and satisfies unforgeability and anonymity, as defined in Section 2.2. We obtain the result in Theorem 3, which we prove via a series of Lemmata.

Theorem 3. *Let the NIZK proof systems for correct credential signing and master secret blinding, and the aggregate NIZK proof system, satisfy the zero-knowledge and soundness properties [14, 43]. Let the Pedersen commitment scheme satisfy the hiding and binding properties [42], and let the CL signature scheme satisfy unforgeability [34]. Then, the* HLAC *protocol satisfies correctness, unforgeability and anonymity.*

Lemma 1 (Credential Correctness). HLAC *satisfies correctness.*

Proof. Consider a credential $\texttt{cred} = (\texttt{ID}_\texttt{S}, \texttt{ID}_\texttt{D}, \texttt{ID}_\texttt{R}, \{(\mathsf{a}_i, v_i)\}_{i\in[L]}, \sigma_p, \rho_\sigma, \mathsf{r}_\texttt{cred}, \texttt{ID}_\texttt{A}, \mathsf{w})$ output by algorithm Issue, and a proof $\texttt{Proof} = (\mathsf{C}, c_H, \{(\rho_i, A_{r,i}, A_{\bar{r},i}, V_{r,i}, \texttt{ID}_{\texttt{S}_i}, \texttt{ID}_{\texttt{D}_i}, \texttt{ID}_{\texttt{R}_i}, \texttt{ID}'_{\texttt{A}_i}, \mathsf{P}'_i)\}_{i\in[|\mathsf{C_P}|]})$ be an identity proof output by algorithm Present in the correctness experiment.

By definition, HLAC satisfies credential correctness if algorithm Store adds cred to list W and algorithm Verify returns 1 with overwhelming probability. We prove by contradiction that these events occur with overwhelming probability, and therefore conclude that HLAC satisfies presentation correctness.

Event 1: cred *is added to list* W.
Assume that algorithm Store does not add cred to W. Then it must be the case that the primary signature on the credential σ does not verify, i.e., algorithm ZK_σ.Verify returns 0. Recall that ZK_σ.Verify, defined in Figure 5, returns 0 if $e \notin [2^{596}, 2^{596}+2119]$, e is not prime, $Q' \neq A^e$ or $c_\sigma \neq H(Q'\|A, \|\tilde{A}'\|n_1)$. Firstly, e is generated by algorithm Issue (and, more specifically, algorithm CL.Sign) such that e is prime and is selected from the given interval. Therefore, the first two conditions are satisfied. We now consider the other two conditions in turn.

Let's assume that $Q' \neq A^e$. From algorithm CL.Sign, $A = Q^{e^{-1} \mod p'q'} \mod n$. Accordingly, $A^e = Q \mod n$. It remains to show that $Q' = Q$.

$$\begin{aligned} Q' &= \frac{Z}{S^v \cdot \prod_{i\in[L]} R_i^{v_i}} \mod n \\ &= \frac{Z}{S^{v'+v''} \cdot R_1^{\texttt{ms}} \cdot \prod_{i\in[2,L]} R_i^{v_i}} \\ &= \frac{Z}{v_1 \cdot S^{v''} \prod_{i\in[2,L]} R_i^{v_i}} \\ &= Q. \end{aligned} \tag{1}$$

Therefore, $Q' = Q$ if all values input to CL.Sign and ZK_σ.Verify are computed correctly.

Finally, we assume that $c_\sigma \neq H(Q'\|A, \|\tilde{A}'\|n_1)$. From algorithm Issue $c_\sigma = H(Q\|A\|\tilde{A}\|n_1)$. Therefore, we require that $Q' \neq Q$ or $\tilde{A}' \neq \tilde{A}$. As per Equation 1, $Q' = Q$. We also have that

$$\begin{aligned} \tilde{A}' &= A^{c_\sigma + s_e \cdot e} \mod n \\ &= A^{c_\sigma + (r - c_\sigma \cdot e^{-1}) \cdot e} \\ &= A^{re} = Q^r = \tilde{A}. \end{aligned} \tag{2}$$

Therefore, $c_\sigma = H(Q'\|A, \|\tilde{A}'\|n_1)$. Then, by contradiction, algorithm Store adds cred to credential list W.

Event 2: Verify *returns* 1.
Assume that algorithm Verify does not return 1. Then we require that $c_H \neq \mathcal{H}(\hat{\mathsf{T}}\|\mathcal{C}\|n_2)$.That is, $\hat{\mathsf{T}} \neq \mathsf{T}$. Recall that $\mathsf{T} = (\mathsf{T}_1, \ldots, \mathsf{T}_{|\mathsf{C_P}|})$ is a tuple generated by algorithm Present where each T_i for $i \in [|\mathsf{C_P}|]$ is a tuple $(\mathsf{T}_{\texttt{cred}_i},$

$\{\mathsf{T}_{\mathsf{pred}_{i,j}}\}_{j\in[|\mathsf{P}'_i|]}, \mathsf{T}_{\mathsf{rev}_i})$ and $\hat{\mathsf{T}}$ is a complementary tuple generated by algorithm Verify. In particular, $\mathsf{T}_{\mathtt{cred}} = \bar{Z}$ is generated for each credential used in the proof by running algorithm $\mathsf{ZK}_{\mathtt{cred}}.\mathsf{Prove1}$; $\mathsf{T}_{\mathsf{pred}} = (\tilde{T}_1, \tilde{T}_2, \tilde{T}_3, \tilde{T}_4, \tilde{T}_\Delta, Q)$ is generated for each predicate by algorithm $\mathsf{ZK}_{\mathsf{pred}}.\mathsf{Prove1}$; and $\mathsf{T}_{\mathsf{rev}} = (\bar{T}_1, \bar{T}_2, \bar{T}_3, \bar{T}_4, \bar{T}_5, \bar{T}_6, \bar{T}_7, \bar{T}_8)$ are values generated for the non-revocation proof by algorithm $\mathsf{ZK}_{\mathsf{rev}}.\mathsf{Prove1}$. Similarly, a verifier running algorithm Verify generates $\hat{\mathsf{T}}$ that consists of tuples generated by running algorithms $\mathsf{ZK}_{\mathtt{cred}}.\mathsf{Verify}$, $\mathsf{ZK}_{\mathsf{pred}}.\mathsf{Verify}$ and $\mathsf{ZK}_{\mathsf{rev}}.\mathsf{Verify}$. We now show that $\hat{\mathsf{T}} = \mathsf{T}$ as required.

$$\begin{aligned}\hat{T} &= \left(\frac{Z}{(\prod_{j\in A_r} R_j^{v_j})(\bar{A})^{2^{596}}}\right)^{-c_H} \cdot (\bar{A})^{\tilde{e}} \cdot \left(\prod_{j\in A_{\bar{r}}} R_j^{\tilde{v_j}}\right) \cdot (S^{\tilde{v}}) \mod n \\ &= \bar{Z} \cdot \left(Z^{-1} \cdot \prod_{j\in[L]} R_j^{v_j} \cdot A^e \cdot S^v\right)^{c_H} \\ &= \bar{Z} \cdot (Q \cdot Q^{-1})^{c_H} = \bar{Z}\end{aligned} \tag{3}$$

$$\begin{aligned}\hat{T}_i &= T_i^{-c_H} \cdot Z^{\hat{u}_i} \cdot S^{\hat{r}_i} \\ &= (Z^{u_i} \cdot S^{r_i})^{-c_H} \cdot Z^{(\tilde{u}_i + c_H \cdot u_i)} \cdot S^{(\tilde{r}_i + c_H \cdot r_i)} = \tilde{T}_i\end{aligned} \tag{4}$$

$$\begin{aligned}\hat{T}_\Delta &= (T_\Delta^a \cdot Z^{\Delta'})^{-c_H} \cdot Z^{\tilde{v}} \cdot S^{a\hat{r}_\Delta} \\ &= (Z^{\Delta \cdot a} \cdot S^{r_\Delta \cdot a} \cdot Z^{\Delta'})^{-c_H} \cdot Z^{\hat{v} + c_H \cdot v} \cdot S^{a \cdot (\tilde{r}_\Delta + c_H \cdot r_\Delta)} \\ &= Z^{\hat{v}} \cdot S^{a \cdot \tilde{r}_\Delta} Z^{(-a \cdot \Delta - \Delta' + v)^{c_H}} \\ &= Z^{\hat{v}} \cdot S^{a \cdot \tilde{r}_\Delta} = \tilde{T}_\Delta\end{aligned} \tag{5}$$

$$\begin{aligned}\hat{Q} &= T_\Delta^{-c_H} \cdot \prod_{i\in[4]} T_i^{\hat{u}_i} \cdot S^{\hat{\alpha}} \mod n \\ &= Z^{-(\Delta \cdot c_H)} \cdot S^{-(c_H \cdot r_\Delta)} \cdot \prod_{i\in[4]} \left(T_i^{(\tilde{u}_i + c_H \cdot u_i)} \cdot S^{(\tilde{\alpha} + c_H \cdot (r_\Delta - \sum_{i\in[4]} u_i \cdot r_i))}\right) \\ &= S^{\tilde{\alpha}} \cdot \prod_{i\in[4]} T_i^{\tilde{u}_i} \cdot Z^{-(\Delta \cdot c_H)} \cdot S^{-(c_H \cdot r_\Delta)} \cdot \prod_{i\in[4]} \left(T_i^{c_H \cdot u_i} \cdot S^{c_H (r_\Delta - \sum_{i\in[4]} u_i r_i)}\right) \\ &= Q\end{aligned} \tag{6}$$

$$\begin{aligned}\hat{T}_1 &= E^{c_H} \cdot h^{\hat{e}} \cdot \tilde{h}^{\hat{o}} \\ &= (h^p \cdot \tilde{h}^o)^{c_H} \cdot h^{\tilde{e} - C_H \cdot p} \cdot \tilde{h}^{\tilde{o} - c_H \cdot o} \\ &= h^{\tilde{e}} \cdot \tilde{h}^{\tilde{o}} = \bar{T}_1\end{aligned} \tag{7}$$

$$\begin{aligned}\hat{T}_2 &= E^{\hat{c}} \cdot h^{-\hat{m}} \cdot \tilde{h}^{-\hat{t}} \\ &= \bar{T}_2 \cdot h^{c_H \cdot (m - c \cdot p)} \cdot \tilde{h}^{c_H \cdot (t - c \cdot o)} = \bar{T}_2\end{aligned} \tag{8}$$

$$\begin{aligned}\hat{T}_3 &= \left(\frac{e(h_0 \cdot G, \hat{h})}{e(A, y)}\right)^{c_H} \cdot e(A, \hat{h})^{\hat{c}} \cdot e(\tilde{h}, \hat{h})^{\hat{r}} \cdot e(\tilde{h}, y)^{-\hat{e}} \\ &\cdot e(\tilde{h}, \hat{h})^{-\hat{m}} \cdot e(h_1, \hat{h})^{\hat{\mathsf{m}}_2} \cdot e(h_2, \hat{h})^{-\hat{s}} \\ &= e(h_0 \cdot G, \hat{h})^{c_H} \cdot e(A, y)^{-c_H} \cdot e(A, \hat{h})^{\tilde{c} - c_H \cdot c} \\ &\cdot e(\tilde{h}, \hat{h})^{\tilde{r} - c_H \cdot r} \cdot e(\tilde{h}, y)^{-\tilde{e} - c_H \cdot p} \cdot e(\tilde{h}, \hat{h})^{-\hat{m} + c_H \cdot m} \\ &\cdot e(h_1, \hat{h})^{-\tilde{\mathsf{m}}_2 + c_H \cdot \mathsf{m}_2} \cdot e(h_2, \hat{h})^{-\tilde{s} + c_H \cdot v_r''} \\ &= \bar{T}_3 \cdot e(h_0 \cdot G, \hat{h})^{c_H} \cdot e(A, y)^{-c_H} \cdot e(A, \hat{h})^{-c_H \cdot c} \\ &\cdot e(\tilde{h}, \hat{h})^{-c_H \cdot r} \cdot e(\tilde{h}, y)^{c_H \cdot p} \cdot e(\tilde{h}, \hat{h})^{c_H \cdot m} \\ &\cdot e(h_1, \hat{h})^{c_H \cdot \mathsf{m}_2} \cdot e(h_2, \hat{h})^{c_H \cdot v_r''} \\ &= \bar{T}_3\end{aligned} \tag{9}$$

$$\begin{aligned}\hat{T}_4 &= \left(\frac{e(G, \mathsf{Acc})}{e(g, W) \cdot z}\right)^{c_H} \cdot e(\tilde{h}, \mathsf{Acc})^{\hat{r}} \cdot e(\frac{1}{g}, \hat{h})^{\hat{r}'} \\ &= e(G, \mathsf{Acc})^{c_H} \cdot e(g, W)^{-c_H} \cdot z^{-c_H} \cdot e(\tilde{h}, \mathsf{Acc})^{\tilde{r} - c_H \cdot r} \cdot e(g, \hat{h})^{-\tilde{r}' + c_H \cdot r'} \\ &= \bar{T}_4 \cdot e(G, \mathsf{Acc})^{c_H} \cdot e(g, W)^{-c_H} \cdot z^{-c_H} \cdot e(\tilde{h}, \mathsf{Acc})^{-c_H \cdot r} \cdot e(g, \hat{h})^{c_H \cdot r'} \\ &= \bar{T}_4\end{aligned} \tag{10}$$

$$\begin{aligned}\hat{T}_5 &= D^{\hat{r}''} \cdot g^{\hat{r}} \cdot \tilde{h}^{\hat{o}'} \\ &= (g^r \cdot \tilde{h}^{o'})^{c_H} \cdot g^{(\tilde{r} - c_H \cdot r)} \cdot \tilde{h}^{(\tilde{o}' - c_H \cdot o')} \\ &= g^{\tilde{r}} \cdot \tilde{h}^{\tilde{o}'} = \bar{T}_5\end{aligned} \tag{11}$$

$$\begin{aligned}\hat{T}_6 &= D^{\hat{r}''} \cdot g^{-\hat{m}'} \cdot \tilde{h}^{-\hat{t}'} \\ &= \bar{T}_6 \cdot g^{-c_H \cdot r \cdot r''} \cdot \tilde{h}^{-c_H \cdot o' \cdot r''} \cdot g^{c_H \cdot m'} \cdot \tilde{h}^{c_H \cdot t'} \\ &= \bar{T}_6\end{aligned} \tag{12}$$

$$\begin{aligned}\hat{T}_7 &= \left(\frac{e(pk \cdot G, S)}{e(g, g')}\right)^{c_H} \cdot (pk \cdot G, \hat{h})^{\hat{r}''} \cdot e(\tilde{h}, \hat{h})^{-\hat{m}'} \cdot e(\tilde{h}, S)^{\hat{r}} \\ &= \bar{T}_7 \cdot e(pk \cdot G, S)^{c_H} \cdot e(g, g')^{-c_H} \\ &\cdot e(pk \cdot G, \hat{h})^{-c_H \cdot r''} \cdot e(\tilde{h}, \hat{h})^{c_H \cdot m'} \cdot e(\tilde{h}, S)^{-c_H \cdot r} \\ &= \bar{T}_7\end{aligned} \tag{13}$$

$$
\begin{aligned}
\hat{T}_8 &= \left(\frac{e(G,u)}{e(g,U)}\right)^{c_H} \cdot e(\tilde{h},u)^{\hat{r}} \cdot e(\frac{1}{g},\hat{h})^{\hat{r}'''} \\
&= \bar{T}_8 \cdot e(g_i,u)^{c_H} \cdot e(\tilde{h},u)^{c_H \cdot r} \cdot e(g,u_i)^{-c_H} \\
&\cdot e(g,\hat{h})^{-c_H \cdot r'''} \cdot e(\tilde{h},u)^{-c_H \cdot r} \cdot e(g,\hat{h})^{c_H \cdot r'''} \\
&= \bar{T}_8
\end{aligned} \tag{14}
$$

Therefore, $\mathsf{T} = \hat{\mathsf{T}}$ as required and the HLAC protocol satisfies correctness.

Lemma 2 (Unforgeability). *Let the aggregate proof system used in algorithm* Present *satisfy soundness [14, 43], the CL signature scheme satisfy unforgeability [34], and let the Pedersen commitment scheme satisfy the binding property [42]. Then,* HLAC *satisfies unforgeability.*

Proof. To prove unforgeability, we define an adversary $\mathcal{A}$ in the unforgeability experiment for the HLAC construction, and show that $\mathcal{A}$ cannot output an identity proof $\texttt{Proof}^*$ for a proof request $\texttt{PReq}^*$ such that the proof verifies. In particular, we show that $\mathcal{A}$ cannot output an identity proof on behalf of a corrupt holder that is not supported by credentials held by the holder, or is a proof output on behalf of a coalition of corrupt holders that no one holder can generate on their own. Indeed, if $\mathcal{A}$ can do this, then $\mathcal{A}$ can produce an aggregate proof without knowledge of a witness and we can therefore construct an adversary that succeeds in the soundness experiment for the aggregate proof. Else, $\mathcal{A}$ can output a valid credential forgery that is used in an identity proof and we can define an adversary that succeeds in the unforgeability experiment for the CL signature scheme. Moreover, we show that $\mathcal{A}$ cannot output a valid forgery of an identity proof on behalf of an honest holder, or we can construct an adversary that succeeds in the soundness experiment for the aggregate proof, or can construct a false master secret that can be used to construct the proof, breaking the binding property of the Pedersen commitment scheme.

More formally, let $\mathcal{A}$ be an adversary that succeeds in the unforgeability experiment for the HLAC construction. By definition, $\mathcal{A}$ succeeds if they output an identity proof $\texttt{Proof}^*$ for a proof request $\texttt{PReq}^*$ such that the proof verifies. We require that the proof request cannot be satisfied by any corrupt holders and that the proof is not the output of oracle $\mathcal{O}$Proof. Accordingly, $\mathcal{A}$ must succeed in one the following ways.

1. $\mathcal{A}$ outputs an identity proof on behalf of a corrupt holder that is not supported by credentials held by the holder (i.e., using revoked credentials or attribute values that differ from those in issued credentials), or is a proof output on behalf of a coalition of corrupt holders that no one holder can generate on their own.
2. $\mathcal{A}$ outputs a valid forgery of an identity proof on behalf of an honest holder.

Let `Success` denote the event that the adversary succeeds in the unforgeability experiment, `Invalid` be the event that the adversary outputs an identity proof

on behalf of a corrupt holder and `Forge` be the event that the adversary generates an identity proof forgery on behalf of an honest holder. Then,

$$\Pr[\texttt{Success}] \leq \Pr[\texttt{Invalid}] + \Pr[\texttt{Forge}].$$

We show that $\Pr[\texttt{Invalid}]$ and $\Pr[\texttt{Forge}]$ are negligible. The result then follows.

Event 1: consider the event `Invalid`. If $\mathcal{A}$ can output an identity proof on behalf of corrupt holders, then we can construct an adversary $\mathcal{A}'$ that succeeds in the soundness experiment for the aggregate proof. Indeed, $\mathcal{A}'$ can simulate the role of the challenger and all oracles to $\mathcal{A}$ in the unforgeability experiment. In particular, $\mathcal{A}'$ can query its proof oracle when $\mathcal{A}$ queries oracle $\mathcal{O}$Proof, and the challenge proof $\texttt{Proof}^*$ output by $\mathcal{A}$ in the experiment is the one input to the adversary $\mathcal{A}'$ in the soundness experiment. Moreover, if $\mathcal{A}$ succeeds in the unforgeability experiment, then $\mathcal{A}'$ can succeed in the soundness experiment.

Else, $\mathcal{A}$ can output a valid forgery of a credential on behalf of an issuer that contains attribute values not signed by the issuer. In this case, we can construct an adversary $\mathcal{A}''$ that succeeds in the unforgeability experiment for the CL signature scheme. Indeed, $\mathcal{A}''$ can simulate the role of the challenger and all oracles to $\mathcal{A}$, and the challenge proof $\texttt{Proof}^*$ output by $\mathcal{A}$ must use a forged credential that adversary $\mathcal{A}''$ can output in the unforgeability experiment against the CL signature scheme. Therefore, by contradiction, $\mathcal{A}$ cannot output an identity proof on behalf of corrupt holders and so $\Pr[\texttt{Invalid}]$ is negligible as required.

Event 2: we now consider the event `Forge`. Then $\mathcal{A}$ generates an identity proof $\texttt{Proof}^*$ forgery on behalf of an honest holder. This means that the adversary does not have access to a credential list and master secret such that the proof is satisfied. Thus, $\mathcal{A}$ must either output an identity proof that breaks the soundness property of the NIZK proof system as above, or $\mathcal{A}$ can prove knowledge of the master secret of an honest holder. We show that, in this event, we can construct an adversary $\mathcal{A}'$ that succeeds in the binding property against the Pedersen commitment scheme. By assumption, the Pedersen commitment scheme satisfies the binding property and, accordingly, $\Pr[\texttt{Forge}]$ is negligible as required.

Lemma 3 (Anonymity). *Let the aggregate proof system satisfy the zero-knowledge property [14, 43] and the Pedersen commitment scheme satisfy the hiding property [42]. Then,* HLAC *satisfies* anonymity.

Proof. To prove anonymity, we define an adversary in the anonymity experiment and proceed through a series of game hops, arriving at a game which is identical regardless of the bit b chosen in the anonymity experiment. We show that the adversary can detect the game hops with negligible probability, and that the adversary in the final game hop can output bit b with probability $\frac{1}{2}$. We briefly describe our game hops here. Firstly, we simulate the NIZK proof of correct master secret blinding when calling oracles $\mathcal{O}$Request and $\mathcal{O}$CredIssuance. This game hop is indistinguishable if the NIZK proof system satisfies the zero-knowledge property. Then, we simulate the aggregate NIZK proof when the adversary queries oracle $\mathcal{O}$Proof, which is indistinguishable if the NIZK proof

system satisfies the zero-knowledge property. Finally, when $b = 1$, we return an identity proof for holder i_0^* in the experiment. This game hop is indistinguishable if the Pedersen commitment scheme satisfies the hiding property. In the final game, the inputs to the adversary are identical for $b = 0$ and $b = 1$ and the result holds.

More formally, let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary in the anonymity experiment for the HLAC construction. We proceed through a series of game hops that we show are indistinguishable to the adversary $\mathcal{A}$. In the final game, the view of $\mathcal{A}$ will be identical for $b = 0$ and $b = 1$. We define Game 0 as the anonymity experiment with bit b chosen uniformly at random. Let S_i be the event that $\mathcal{A}$ correctly guesses b in Game i.

Game 1: Game 1 is identical to game 0 except that, when $\mathcal{A}$ queries oracles $\mathcal{O}$Request or $\mathcal{O}$CredIssuance, we simulate the NIZK proof of correct master secret blinding. Game 0 and Game 1 are indistinguishable if the NIZK proof system for correct master secret blinding satisfies the zero-knowledge property. Indeed, let $\mathcal{D}_1$ be a distinguishing algorithm that attempts to output a bit b' in the zero-knowledge experiment. If $b' = 0$, $\mathcal{D}_1$ is provided with a real zero-knowledge proof as in Game 0. If $b' = 1$ $\mathcal{D}_1$ is provided with a simulated zero-knowledge proof as in Game 1. Note that, with the exception of queries to oracles $\mathcal{O}$Request or $\mathcal{O}$CredIssuance, the view of $\mathcal{A}$ is identical in Games 0 and 1. Moreover, if $b' = 0$ (resp., $b' = 1$), inputs to $\mathcal{A}$ are identical to Game 0 (resp., Game 1). Therefore,

$$|\Pr[S_0] - \Pr[S_1]| \leq \mathsf{negl}_1.$$

Game 2: Game 2 is identical to game 1 except that, when $\mathcal{A}$ queries oracle $\mathcal{O}$Proof, we simulate the aggregate NIZK proof for the identity proof. Furthermore, we simulate the challenge aggregate NIZK proof input to $\mathcal{A}_2$ in the experiment. Game 1 and Game 2 are indistinguishable if the aggregate NIZK proof system satisfies the zero-knowledge property. Indeed, let $\mathcal{D}_2$ be a distinguishing algorithm that attempts to output a bit b' in the zero-knowledge experiment. As in the previous game hop, if $b' = 0$, $\mathcal{D}$ is provided with a real zero-knowledge proof and, if $b' = 1$, $\mathcal{D}$ is provided with a simulated zero-knowledge proof. The view of $\mathcal{A}$ is identical in Games 0, except for queries to oracle $\mathcal{O}$Proof and the challenge identity proof. Moreover, if $b' = 0$ (resp., $b' = 1$), inputs to $\mathcal{A}$ are identical to Game 0 (resp., Game 1). Therefore,

$$|\Pr[S_1] - \Pr[S_2]| \leq \mathsf{negl}_2.$$

Game 3: Game 3 is identical to game 2 but, when $b = 1$, the experiment computes the challenge identity proof as $\mathtt{Proof}^* \leftarrow\!\$\, \mathsf{Present}(\mathtt{PReq}^*, \mathbf{L}[i_0^*], \mathbf{H}[i_0^*])$. That is, regardless of b, the experiment always returns an identity proof for holder i_0^*. As holders i_0^* and i_1^* both have the same attribute values and use the same credentials in the proof, the only distinguishing feature is the blinded master secret used in the proof. Therefore, we show that Game 2 and Game 3 are indistinguishable if the Pedersen commitment scheme satisfies the hiding property. Let $\mathcal{D}_3$ be a distinguishing algorithm that attempts to output a bit b' in the hiding experiment. If $b' = 0$ and $b = 1$, $\mathcal{D}_3$ is provided with a commitment

for a message m_0 (i.e., the master secret of holder i_0^*) as in Game 3. If $b' = 1$ and $b = 1$, $\mathcal{D}_3$ is provided with a commitment to a message m_1 (i.e., the master secret of holder i_1^*) as in Game 2. Note that, with the exception of the challenge identity proof, the view of $\mathcal{A}$ is identical in Games 2 and 3. Moreover, if $b' = 0$ (resp., $b' = 1$), inputs to $\mathcal{A}$ are identical to Game 3 (resp., Game 2). Therefore,

$$|\Pr[S_2] - \Pr[S_3]| \leq \mathsf{negl}_3.$$

In Game 3, the inputs to $\mathcal{A}$ are identical for $b = 0$ and $b = 1$. Therefore $\Pr[S_3] = \frac{1}{2}$.

We have that,

$$\Pr\left[S_0 - \frac{1}{2}\right] \leq \mathsf{negl}_1 + \mathsf{negl}_2 + \mathsf{negl}_3$$

which is negligible as required. We conclude that HLAC satisfies anonymity.

6 Concluding Remarks

In this paper, we presented a first formal description and analysis of the Hyperledger AnonCreds protocol specified in [26]. We introduced a security model for the protocol that captures notions of unforgeability and anonymity from the anonymous credentials literature. As our work is a first security analysis of the HLAC protocol, we believe that our findings provide valuable insight into the protocol's security, and can be used as a springboard for future research. We now conclude by discussing the challenges of analysing the HLAC protocol, and highlight potential extensions to our modelling.

Following completion of the HLAC specification, work will begin on v2.0 of Hyperledger AnonCreds, which uses Boneh-Boyen-Shachum (BBS) signatures [13] in place of CL signatures. A comparison of these two approaches and security analysis of Hyperledger AnonCreds v2.0 are potential directions for future work. We are confident that our syntax and security model will be straightforward to adapt to v2.0 of the specification. However, because the signatures and proofs used in the BBS construction vary widely from CL signatures, the protocol description and security proofs presented in this paper will likely need fairly significant changes. We therefore consider an extension to v2.0 of the specification to be a substantial and important piece of future work.

One of the greatest challenges in formal analysis of real-world protocols is accurately capturing the design of the protocol in the formal modelling. In this respect, specification of a protocol is invaluable. Though the HLAC specification [26] provides an overview of the protocol and link to a reference implementation, the specification is, at the time of writing, incomplete. A complete specification will allow us to verify our findings. We see it as potential future work to modify our modelling to capture any future changes to the HLAC specification.

Acknowledgment

We are grateful to the anonymous reviewers for comments on an earlier draft of this paper. This work was started when the first author was at University of Surrey. Both authors are grateful to EPSRC for funding this work under the DECaDE project P/T022485/1.

References

1. Tolga Acar and Lan Nguyen. Revocation for delegatable anonymous credentials. In *14th IACR International Conference on Practice and Theory of Public Key Cryptography*, pages 423–440. Springer, 2011.
2. Michael Backes, Lucjan Hanzlik, Kamil Kluczniak, and Jonas Schneider. Signatures with flexible public key: Introducing equivalence classes for public keys. In *ASIACRYPT 2018: 24th Annual International Conference on the Theory and Application of Cryptology and Information Security*, pages 405–434. Springer, 2018.
3. Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT 1997: 15th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 480–494. Springer, 1997.
4. Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In *CRYPTO 2009: 29th Annual International Cryptology Conference*, pages 108–125. Springer, 2009.
5. Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. Non-interactive anonymous credentials. Cryptology ePrint Archive, Paper 2007/384, 2007.
6. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
7. Josh Cohen Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In *EUROCRYPT 1993: Workshop on the Theory and Application of of Cryptographic Techniques*, pages 274–285. Springer, 1993.
8. Magdalena Bertram, Marian Margraf, Maximilian Richter, and Jasper Seidensticker. Analysis of the anonymous credential protocol 'anoncreds 1.0' used in hyperledger indy, 2022.
9. Johannes Blömer and Jan Bobolz. Delegatable attribute-based anonymous credentials from dynamically malleable signatures. In *37th Annual International Conference on Applied Cryptography and Network Security (ACNS 2018)*, pages 221–239. Springer, 2018.
10. Johannes Blömer, Jan Bobolz, Denis Diemert, and Fabian Eidens. Updatable anonymous credentials and applications to incentive systems. In *Proceedings of the 2019 ACM Conference on Computer and Communications Security*, pages 1671–1685, 2019.
11. Jan Bobolz, Fabian Eidens, Stephan Krenn, Sebastian Ramacher, and Kai Samelin. Issuer-hiding attribute-based credentials. In *Cryptology and Network Security: 20th International Conference (CANS 2021)*, pages 158–178. Springer, 2021.

12. Sharon Boeyen, Stefan Santesson, Tim Polk, Russ Housley, Stephen Farrell, and David Cooper. RFC 5280: Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile, May 2008.
13. Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO 2004: 24th Annual International Cryptology Conference*, pages 41–55. Springer, 2004.
14. Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In *12th International Conference on Practice and Theory of Public Key Cryptography*, pages 481–500. Springer, 2009.
15. Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. Solving revocation with efficient update of anonymous credentials. In *Security and Cryptography for Networks*, pages 454–471. Springer Berlin Heidelberg, 2010.
16. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT 2001: 20th Annual International Conference on the Theory and Application of Cryptographic Techniques*, pages 93–118. Springer, 2001.
17. Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO 2002: 22nd Annual International Cryptology Conference*, pages 61–76. Springer, 2002.
18. Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In *International Conference on Security in Communication Networks*, pages 268–289. Springer, 2002.
19. Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO 2004: 24th Annual International Cryptology Conference*, pages 56–72. Springer, 2004.
20. Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 21–30, 2002.
21. Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In *CRYPTO 2006: 26th Annual International Cryptology Conference*, pages 78–96. Springer, 2006.
22. Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic macs and keyed-verification anonymous credentials. In *Proceedings of the 2014 ACM Conference on Computer and Communications Security*, pages 1205–1216, 2014.
23. David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.
24. Aisling Connolly, Pascal Lafourcade, and Octavio Perez Kempner. Improved constructions of anonymous credentials from structure-preserving signatures on equivalence classes. In *25th International Conference on Practice and Theory of Public Key Cryptography*, pages 409–438. Springer, 2022.
25. Elizabeth C Crites and Anna Lysyanskaya. Delegatable anonymous credentials from mercurial signatures. In *Topics in Cryptology–CT-RSA 2019: The Cryptographers' Track at the RSA Conference 2019*, pages 535–555. Springer, 2019.
26. Stephen Curran, Artur Philipp, Hakan Yildiz, Sam Curren, Victor Martinez Jurado, Aritra Bhaduri, and Artem Ivanov. Anoncreds specification. `https://hyperledger.github.io/anoncreds-spec/`, 2022.
27. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *EUROCRYPT 1986: Workshop on the Theory and Application of Cryptographic Techniques*, pages 186–194. Springer, 1986.

28. Hyperledger Foundation. Hyperledger Foundation Hyperledger Aries. `https://www.hyperledger.org/use/aries`.
29. Hyperledger Foundation. Hyperledger Foundation Hyperledger Indy. `https://www.hyperledger.org/use/hyperledger-indy`.
30. Linux Foundation. Decentralized Trust. `https://www.lfdecentralizedtrust.org/`.
31. Georg Fuchsbauer. Commuting signatures and verifiable encryption. In *EUROCRYPT 2011: 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 224–245. Springer, 2011.
32. Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. *Journal of Cryptology*, 32:498–546, 2019.
33. Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO'97: 17th Annual International Cryptology Conference*, pages 16–30. Springer, 1997.
34. Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.
35. Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for np. In *EUROCRYPT 2006: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 339–358. Springer, 2006.
36. Lucjan Hanzlik and Daniel Slamanig. With a little help from my friends: Constructing practical anonymous credentials. In *Proceedings of the 2021 ACM Conference on Computer and Communications Security*, pages 2004–2023, 2021.
37. IBM. Idemix Anonymous Identity Stack Implementation. `https://github.com/IBM/idemix`.
38. Nesrine Kaaniche, Maryline Laurent, Pierre-Olivier Rocher, Christophe Kiennert, and Joaquin Garcia-Alfaro. A privacy-preserving certification scheme. In *International Workshop on Data Privacy Management*, pages 239–256. Springer, 2017.
39. Saqib A Kakvi, Keith M Martin, Colin Putman, and Elizabeth A Quaglia. Sok: Anonymous credentials. In *International Conference on Research in Security Standardisation*, pages 129–151. Springer, 2023.
40. Government of British Columbia. OrgBook. `https://www.orgbook.gov.bc.ca/about`.
41. Ilker Ozcelik., Sai Medury., Justin Broaddus., and Anthony Skjellum. An overview of cryptographic accumulators. In *Proceedings of the 7th International Conference on Information Systems Security and Privacy - ICISSP*, pages 661–669. SciTePress, 2021.
42. Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO 1991: Annual International Cryptology Conference*, pages 129–140. Springer, Berlin, Heidelberg, 1991.
43. IBM Research. Specification of the Identity Mixer Cryptographic Library Version 2.3.0. `https://dominoweb.draco.res.ibm.com/reports/rz3730_revised.pdf`.
44. Olivier Sanders. Efficient redactable signature and application to anonymous credentials. In *23rd IACR International Conference on Practice and Theory of Public Key Cryptography*, pages 628–656. Springer, 2020.
45. Siamak F Shahandashti and Reihaneh Safavi-Naini. Threshold attribute-based signatures and their application to anonymous credential systems. In *AFRICACRYPT 2009: 2nd International Conference on Cryptology in Africa*, pages 198–216. Springer, 2009.

46. Manu Sporny, Dave Longley, and David Chadwick. Verifiable credentials data model 1.0. https://www.w3.org/TR/vc-data-model/.
47. Linux Foundation Decentralized Trust. Hyperledger AnonCreds Working Group. https://lf-hyperledger.atlassian.net/wiki/spaces/ANONCREDS/pages/20291468/AnonCreds+Working+Group.
48. Eric R Verheul. Self-blindable credential certificates from the weil pairing. In *ASIACRYPT 2001: 7th Annual International Conference on the Theory and Application of Cryptology and Information Security*, pages 533–551. Springer, 2001.
49. Yan Zhang and Dengguo Feng. Efficient attribute proofs in anonymous credential using attribute-based cryptography. In *14th International Conference on Information and Communications Security, ICICS 2012*, pages 408–415. Springer, 2012.