Analysis of the Anonymous Credential Protocol 'AnonCreds 1.0' used in Hyperledger Indy

Fraunhofer AISEC
Magdalena Bertram, Marian Margraf,
Maximilian Richter, Jasper Seidensticker

March 27, 2022

Contents

Part I
Introduction

In this report, we will take a look at the AnonCreds 1.0 protocol as introduced in [KL05], which is the mathematical foundation for the anonymous credential protocol implemented in Hyperledger Indy. The main structure of AnonCreds 1.0 is based on the Idemix specification [Res10] augmented with a different approach for revocations, namely a type-3 pairing version of the accumulator mechanism presented by Camenisch et al. [CKS08].

A key part in the anonymous credential protocol used in Hyperledger Indy is the Camenisch-Lysyanskaya (CL) signature scheme, which was published in 2002 [CL02].

The goal of this document is to give a formal description of the protocol and to describe some of its security properties. Some additional mathematical foundations and explanations can be found in the appendix.

Part II
AnonCreds 1.0

1 System overview

Anonymous credential protocols enable users to prove statements about their identity without revealing any additional information. As an example, a proof of a person having reached the legal age or living in a certain area without revealing their actual age or their full address is possible. In addition, multiple proofs by the same holder should not be linkable. Therefore, a holder never reveals concrete values, not even credential signatures. Instead, zero-knowledge proofs are applied for verification.

In the system at hand we have three entities, namely issuer, holder and verifier. Continuing our example, the holder is the legal person, the issuer is the authority providing identification and the verifier is any entity entitled to check certain attributes. The relationship between the entities is also shown in Figure 1.

Note that the system may consist of a variety of issuers leaving a person with potentially many different credentials. This is where the properties of blockchain comes in handy. As seen in Figure 1, the blockchain stores identifiers of the participating Indy nodes (DIDs), public keys of all issuers, credential schemas as well as revocation data.

2 CL signature scheme

In this section, we describe the Camenisch-Lysyanskaya (CL) signature scheme as presented in [CG08]. CL signatures are particularly useful in a 3-party scenario like we have in AnonCreds. They allow a user to efficiently obtain a signature on committed messages from a signer. They furthermore allow the user to convince a verifier that he possesses a signature by the signer on a committed message.

Analogously to RSA, the key generation produces a module n consisting of two safe prime numbers p and q. In addition, we need $L + 2$ samples from the set of quadratic residues QR_n, which form the public key.

[1]This illustration is taken from "Systemkonzept zu dem Projekt 2080069021: Digitale Identitäten: Pilotvorhaben Hotel Check-In" by IBM Deutschland.

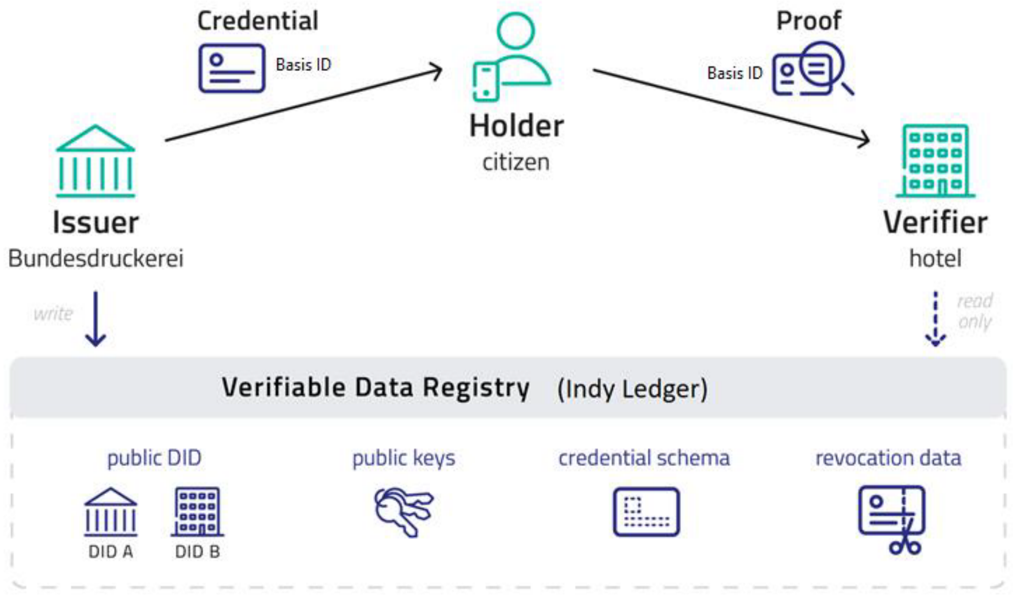

Figure 1: AnonCreds 1.0 System Overview[1]

Algorithm 1: CL Key Generation

```
Input: none
    1. Sample sufficiently large safe primes $p, q$
    2. Calculate $n = p \cdot q$
    3. Sample $R_1, \dots, R_L, S, Z \in QR_n$
Output: public key $pk = (n, R_1, \dots, R_L, S, Z)$, secret key $sk = (p, q)$
```

Let $m_1, \dots, m_L$ be a set of messages, e.g. attributes of an identification card. A valid signature is a tuple (A, e, v) fulfilling

$$A^e = \frac{Z}{R_1^{m_1} \cdot \ldots \cdot R_L^{m_L} \cdot S^v} \mod n.$$

As in the case of RSA, the value A is not directly calculated as an e-th root (which is computationally hard), but instead via a corresponding value $d \in \mathbb{Z}_n$ such that $(A^e)^d = A$ holds for all $A \in \mathbb{Z}_n$. With knowledge of the factorization of n, d can be efficiently calculated using the extended Euclidean algorithm.

While the values e and v are sampled, the corresponding value A can be calculated with knowledge of the public key as well as the factors $R_i^{m_i}$. Here, we can see one of the more interesting properties of CL signatures: The signer, i.e. the issuer in our scenario, does in fact not need to know all the plaintext values $\{m_i\}_i$. Instead, the holder can pre-compute the value $\prod_i R_i^{m_i} \cdot S^{v'}$ instead of the individual $\{m_i\}_i$, thereby hiding these particular attributes from the signer. The additional factor $S^{v'}$ is randomly sampled and thereby prevents dictionary attacks. We call $R_i^{m_i}$ the *committed* form of m_i.

Algorithm 2: CL Signature Generation

```
Input: $(m_1 \text{ or } R_1^{m_1}), \dots, (m_L \text{ or } R_L^{m_L}), sk = (p, q), pk = (n, R_1, \dots, R_L, S, Z)$
    1. Sample prime $e \in \mathbb{Z}_n$ with $e \neq p, q$
    2. Sample $v \in \mathbb{Z}_n$
    3. Calculate $d$ via extended Euclidean algorithm
    4. Calculate $A = (\frac{Z}{R_1^{m_1} \cdot \ldots \cdot R_L^{m_L} \cdot S^v})^d \mod n$
Output: signature $\sigma = (A, e, v)$
```

Signature verification is straight-forward. A verifier can validate a given signature with knowledge of the public key and either m_i or $R_i^{m_i}$ for every $1 \leq i \leq L$. This enables the holder to provide just the specific attributes he wants. To summarize, CL signatures can be verified without knowing a single plaintext m_i.

Algorithm 3: CL Signature Verification

```
Input: $(m_1 \text{ or } R_1^{m_1}), \ldots, (m_L \text{ or } R_L^{m_L})$, $\sigma = (A, e, v), pk = (n, R_1, \ldots, R_L, S, Z)$
  1. Calculate $Z' = A^e \cdot R_1^{m_1} \cdot \ldots \cdot R_L^{m_L} \cdot S^v \mod n$
Output: valid if $Z = Z'$, else invalid
```

Requirements on the bit length of relevant variables can be found in the original CL signature introduction paper [CL02].

3 Credentials

Credentials are digital identities consisting of a set of attributes. In AnonCreds 1.0, we also want the credentials to be anonymous, i.e. having the following properties:

- Correctness: An honest user must always succeed in constructing correct proofs for the verifier.
- Anonymity: The verifier cannot learn anything else about the presented credentials except the attributes chosen by the holder, i.e. the holder can selectively disclose certain attributes.
- Unforgeability: The verifier can reliably validate whether a given credential was signed by the issuer.
- Unlinkability: Different usages of the same credential should not be linkable, i.e. a verifier should not be able to detect correlations between them.

We present the generic credential schema in Table 1 consisting of some technical IDs, the actual holder attributes as well as the corresponding CL signature.

The attribute m_1 is used to prove that different credentials belong to one person, thereby linking multiple credentials (see Section 6). The attribute m_2 can be used by the issuer to revoke the credential (see Section 7).

m_1:	Private ID (Link Secret)
m_2:	Credential ID (Revocation)
m_3:	Attribute 3
m_4:	Attribute 4
m_5:	Attribute 5
$\vdots$	$\vdots$
m_L:	Attribute L
CL-Signatur:	(A, e, v)

Table 1: Credential schema used in AnonCreds 1.0

4 Credential issuing

To enable the signing of credentials, the issuer initially needs to generate a long-term key pair $(PK_{issuer}, SK_{issuer})$, which is used to generate credentials for all holders. This is done via the CL key generation routine as described in Algorithm 1.

The public key PK_{issuer} is published on the ledger while the corresponding private key SK_{issuer} is locally stored with the issuer. The issuer also determines and publishes a credential schema $\mathcal{S}$, i.e. an assignment of attributes to the variables m_i.

4.1 Abstract overview

On a very high level, a holder planning to request a credential retrieves the credential schema $\mathcal{S}$, sets the hidden attributes $\mathcal{M}_{hidden}$ and sends the committed hidden attributes as well as all the other non-committed attributes $\mathcal{M}_{revealed}$ to the issuer.

If the issuer agrees with the provided attributes, he builds and signs a credential via the CL signature generation routine as described in Algorithm 2 with SK_{issuer}.

Holder		**Issuer**
	$\xleftarrow{\mathcal{S},\ PK_{issuer}}$	
Set: $\mathcal{M}_{hidden}$		
	$\xrightarrow{\text{committed } \mathcal{M}_{hidden}}$	
	$\xleftrightarrow{\text{agree on } \mathcal{M}_{revealed}}$	
		Build: $\mathcal{C}'_p$, sign with SK_{issuer}
	$\xleftarrow{\mathcal{C}'_p}$	
Compute: $\mathcal{C}_p$ from $\mathcal{C}'_p$ Store: $\mathcal{C}_p$ in wallet		

Table 2: Abstract credential issuing protocol

The detailed credential issuing protocol is presented in the following section.

4.2 Issuing of credentials

- $n, S, Z, \{R_i\}$: Issuer's public key stored on public ledger
- $p = 2p' + 1, q = 2q' + 1$: Issuer's private key
- $\mathcal{S}$: Credential schema determined by issuer
- A_h: Index set of hidden attributes as specified in $\mathcal{S}$
- A_r: Index set of revealed attributes as specified in $\mathcal{S}$
- v': Random value sampled by holder
- v'', e: Random values sampled by issuer
- $C_p = (\{m_i\}, A, e, v)$: Holder's credential

Holder		**Issuer**
	$\xleftarrow{\mathcal{S},\ PK_{issuer}}$	
Set: $\{m_i\}_{i \in A_h}$		
Sample: v'		
Compute: $U = S^{v'} \cdot \left(\prod_{i \in A_h} R_i^{m_i} \right) \mod n$ Proof $\mathcal{P}_U$		
	$\xrightarrow{U, \mathcal{P}_U}$	
		Verify: $\mathcal{P}_U$
		Set: $\{m_i\}_{i \in A_r}$
		Sample: v'', e
		Compute: $Q = \frac{Z}{U \cdot S^{v''} \cdot \left(\prod_{i \in A_r} R_i^{m_i} \right)} \mod n$ $d = e^{-1} \mod p'q'$ $A = Q^d \mod n$ Proof $\mathcal{P}_A$
	$\xleftarrow{\{m_i\}_{i \in A_r}, A, e, v'', \mathcal{P}_A}$	
Compute: $v = v' + v''$ $Q = \frac{Z}{S^v \cdot \left(\prod_{i \in (A_h \cup A_r)} R_i^{m_i} \right)} \mod n$		
Verify: $\mathcal{P}_A$		
Verify: $Q = A^e \mod n$		
Store: $C_p = (\{m_i\}, A, e, v)$		

Table 3: Credential issuing protocol

The proof $\mathcal{P}_U$ in Table 3 is a zero-knowledge proof that the value U was indeed constructed according to the protocol, i.e. the holder does in fact have knowledge of all the contained variables $\{m_i\}_{i \in A_h}$. The proof $\mathcal{P}_A$ is a zero-knowledge proof that the verifier also acted according to the protocol.

5 Credential verification

5.1 Abstract overview

On a very high level, the holder retrieves the credential schema $\mathcal{S}$ from the verifier as well as the required attribute predicates $\mathcal{D}$, i.e. requirements on a subset of the contained attributes. For example, the age value could be required to be at least a certain value.

The holder prepares a zero-knowledge proof showing that the credential is valid and satisfies the demanded attribute predicates $\mathcal{D}$.

Holder		**Verifier**
	$\xleftarrow{\mathcal{S},\ \mathcal{D}}$	
Create proof $\mathcal{P}$		
	$\xrightarrow{\mathcal{P}}$	
		Verify $\mathcal{P}$

Table 4: Abstract credential verification protocol

The actual proof preparation and presentation description is presented in the following sections.

5.2 Verification of revealed attributes and signature correctness

This section presents the verification of revealed attributes, i.e. attributes which are actively disclosed by the holder. For now, we restrict this scenario to a single credential schema $\mathcal{S}_1$. The extension to multiple credential schemata is presented later in section 6.

Relevant variables:

- $C_p = (\{m_i\}, A, e, v)$: Holder's credential
- $n, S, Z, \{R_i\}$: Issuer's public key stored on public ledger
- $\mathcal{S}_1$: Credential schema requested by the verifier
- A_h: Index set of hidden attributes as specified in $\mathcal{S}_1$
- A_r: Index set of revealed attributes as specified in $\mathcal{S}_1$
- n_1: Nonce sampled by verifier
- $\mathcal{T}, \mathcal{C}$: Initially empty sets used to collect various values
- $\{\widetilde{m_i}\}$: Random values obscuring hidden attributes; to prove that multiple hidden attributes are equal without disclosing them, set them to the same random value (e.g. for credential linking)
- $\widetilde{e}, \widetilde{v}, r$: Random values sampled by holder

Holder		**Verifier**
		Sample: n_1
	$\xleftarrow{\mathcal{S}_1, A_h, A_r, n_1}$	
Sample: $\{\widetilde{m_j}\}_{j \in A_h}, \widetilde{e}, \widetilde{v}, r$		
Compute: $A' = A \cdot S^r \mod n$ $e' = e - 2^{596}$ $v' = v - e \cdot r$		
Add to $\mathcal{C}$: A', v'		
Compute: $T = (A')^{\widetilde{e}} \cdot \left(\prod_{j \in A_h} R_j^{\widetilde{m}_j} \right) \cdot S^{\widetilde{v}} \mod n$		
Add to $\mathcal{T}$: T		
Compute: $c_H = Hash(\mathcal{T}, \mathcal{C}, n_1)$		
Compute: $\widehat{e} = \widetilde{e} + c_H \cdot e'$ $\widehat{v} = \widetilde{v} + c_H \cdot v'$ $\{\widehat{m_j} = \widetilde{m_j} + c_H \cdot m_j\}$		
	$\xrightarrow{\{m_i\}_{i \in A_r}, \mathcal{C}, c_H, \widehat{e}, \widehat{v}, \{\widehat{m_j}\}}$	
		Compute: $\widehat{T}$ as specified below Add to $\widehat{\mathcal{T}}$: $\widehat{T}$ $\widehat{c}_H = Hash(\widehat{\mathcal{T}}, \mathcal{C}, n_1)$
		Compare: $\widehat{c}_H$, c_H

Table 5: Verification of revealed attributes and signature correctness

Table 5 shows the verification routine between holder and verifier. The omitted calculation of the variable $\widehat{T}$ is presented in the following equation.

$$
\begin{aligned}
\widehat{T} &= \left(\frac{Z}{\prod_{j \in A_r} R_j^{m_j} \cdot (A')^{2^{596}}}\right)^{-c_H} \cdot (A')^{\widehat{e}} \cdot \left(\prod_{j \in A_h} R_j^{\widehat{m_j}}\right) \cdot S^{\widehat{v}} \\
&= \left(\frac{\prod_{j \in A_r} R_j^{m_j} \cdot (A')^{2^{596}}}{Z}\right)^{c_H} \cdot (A')^{\widehat{e}} \cdot \left(\prod_{j \in A_h} R_j^{\widehat{m_j}}\right) \cdot S^{\widehat{v}} \\
&= \left(\frac{\prod_{j \in A_r} R_j^{m_j} \cdot (A')^{2^{596}}}{Z}\right)^{c_H} \cdot (A')^{\widetilde{e} + c_H \cdot e'} \cdot \left(\prod_{j \in A_h} R_j^{\widehat{m_j}}\right) \cdot S^{\widetilde{v} + c_H \cdot v'} \\
&= \left(\frac{\prod_{j \in A_r} R_j^{m_j} \cdot (A')^{2^{596}}}{Z}\right)^{c_H} \cdot (A')^{\widetilde{e} + c_H \cdot (e - 2^{596})} \cdot \left(\prod_{j \in A_h} R_j^{\widehat{m_j}}\right) \cdot S^{\widetilde{v} + c_H \cdot (v - e \cdot r)} \\
&= \left(\frac{\prod_{j \in A_r} R_j^{m_j}}{Z}\right)^{c_H} \cdot (A')^{\widetilde{e} + c_H \cdot e} \cdot \left(\prod_{j \in A_h} R_j^{\widehat{m_j}}\right) \cdot S^{\widetilde{v} + c_H \cdot (v - e \cdot r)} \\
&= \left(\frac{\prod_{j \in A_r} R_j^{m_j}}{Z}\right)^{c_H} \cdot (A')^{\widetilde{e}} \cdot (A \cdot S^r)^{c_H \cdot e} \cdot \left(\prod_{j \in A_h} R_j^{\widehat{m_j}}\right) \cdot S^{\widetilde{v} + c_H \cdot (v - e \cdot r)} \\
&= \left(\frac{\prod_{j \in A_r} R_j^{m_j}}{Z}\right)^{c_H} \cdot (A')^{\widetilde{e}} \cdot (A^e)^{c_H} \cdot S^{r \cdot c_H \cdot e} \cdot \left(\prod_{j \in A_h} R_j^{\widehat{m_j}}\right) \cdot S^{\widetilde{v} + c_H \cdot (v - e \cdot r)} \\
&= \left(\frac{\prod_{j \in A_r} R_j^{m_j}}{Z}\right)^{c_H} \cdot (A')^{\widetilde{e}} \cdot \left(\frac{Z}{\prod_{j \in A_h \cup A_r} R_j^{m_j} \cdot S^v}\right)^{c_H} \cdot S^{r \cdot c_H \cdot e} \cdot \left(\prod_{j \in A_h} R_j^{\widehat{m_j}}\right) \cdot S^{\widetilde{v} + c_H \cdot (v - e \cdot r)} \\
&= \left(\frac{\prod_{j \in A_r} R_j^{m_j}}{\prod_{j \in A_h \cup A_r} R_j^{m_j}}\right)^{c_H} \cdot (A')^{\widetilde{e}} \cdot S^{-v \cdot c_H} \cdot S^{r \cdot c_H \cdot e} \cdot \left(\prod_{j \in A_h} R_j^{\widehat{m_j}}\right) \cdot S^{\widetilde{v} + c_H \cdot (v - e \cdot r)} \\
&= \left(\frac{\prod_{j \in A_r} R_j^{m_j}}{\prod_{j \in A_h \cup A_r} R_j^{m_j}}\right)^{c_H} \cdot (A')^{\widetilde{e}} \cdot \left(\prod_{j \in A_h} R_j^{\widehat{m_j}}\right) \cdot S^{\widetilde{v}} \\
&= \left(\frac{1}{\prod_{j \in A_h} R_j^{m_j}}\right)^{c_H} \cdot (A')^{\widetilde{e}} \cdot \left(\prod_{j \in A_h} R_j^{\widetilde{m}_j + c_H \cdot m_j}\right) \cdot S^{\widetilde{v}} \\
&= (A')^{\widetilde{e}} \cdot \left(\prod_{j \in A_h} R_j^{\widetilde{m_j}}\right) \cdot S^{\widetilde{v}} \\
&= T
\end{aligned}
$$

As seen in the equation above, the calculated value $\widehat{T}$ indeed matches with T if the tuple of sent values was correctly generated and is based on a valid CL signature (A, e, v). In this case, the calculated hash c_H matches as well. Note that the verifier only gains information on the revealed attributes $\{m_j\}_{j \in A_r}$ while the remaining attributes as well as the actual CL signature stay hidden.

5.3 Verification of required predicates

This section presents the verification process which checks whether the holder's credential C_p satisfies the verifier's predicate set $\mathcal{D}$. Note that this is usually done without actually disclosing the concrete attributes.

Relevant variables:

- $C_p = (\{m_i\}, A, e, v)$: Holder's credential
- $n, S, Z, \{R_i\}$: Issuer's public key stored on public ledger
- S_1: Schema requested by the verifier
- $\{\widetilde{m}_j\}, \{\widehat{m}_j\}, n_1$: Values from the previous protocol
- $\{r_i\}, r_\Delta, \{\widetilde{u}_i\}, \{\widetilde{r}_i\}, \widetilde{r}_\Delta, \widetilde{\alpha}$: Random values sampled by holder

Holder **Verifier**

For each predicate p in $\mathcal{D}$:

Compute: (Δ, a) as specified below this figure
Find decomposition:
$u_1^2 + u_2^2 + u_3^2 + u_4^2 = \Delta$

Sample:
$r_1, r_2, r_3, r_4, r_\Delta$

Compute:
$\{T_i = Z^{u_i} \cdot S^{r_i} \mod n\}$
$T_\Delta = Z^{\Delta} \cdot S^{r_\Delta} \mod n$

Add to $\mathcal{C}$: $\{T_i\}, T_\Delta$

Sample:
$\{\widetilde{u}_i\}, \{\widetilde{r}_i\}, \widetilde{r}_\Delta, \widetilde{\alpha}$

Compute:
$\{\overline{T}_i = Z^{\widetilde{u}_i} \cdot S^{\widetilde{r}_i} \mod n\}$
$\overline{T}_\Delta = Z^{\widetilde{m}_j} \cdot S^{a \cdot \widetilde{r}_\Delta} \mod n$
$Q = S^{\widetilde{\alpha}} \cdot \prod_{i=1}^{4} T_i^{\widetilde{u}_i} \mod n$

Add to $\mathcal{T}$: $\{\overline{T}_i\}, \overline{T}_\Delta, Q$

Compute:
$c_H = Hash(\mathcal{T}, \mathcal{C}, n_1)$

For each predicate p in $\mathcal{D}$:

Compute:
$\{\widehat{u}_i = \widetilde{u}_i + c_H \cdot u_i\}$
$\{\widehat{r}_i = \widetilde{r}_i + c_H \cdot r_i\}$
$\widehat{r}_\Delta = \widetilde{r}_\Delta + c_H \cdot r_\Delta$
$\widehat{\alpha} = \widetilde{\alpha} + c_H \cdot (r_\Delta - u_1 r_1 - u_2 r_2 - u_3 r_3 - u_4 r_4)$

Set sub-proof: $Pr_p = (\{\widehat{u}_i\}, \{\widehat{r}_i\}, \widehat{r}_\Delta, \widehat{\alpha}, \widehat{m}_j)$

$$\xrightarrow{\mathcal{C}, c_H, \{Pr_p\}}$$

For each predicate p in $\mathcal{D}$:

Compute: (Δ', a) as specified below

Compute:
$\{\widehat{T}_i = T_i^{-c_H} \cdot Z^{\widehat{u}_i} \cdot S^{\widehat{r}_i} \mod n\}$
$\widehat{T}_\Delta = \left(T_\Delta^a \cdot Z^{\Delta'}\right)^{-c_H} \cdot Z^{\widehat{m}_j} \cdot S^{a \cdot \widehat{r}_\Delta} \mod n$
$\widehat{Q} = T_\Delta^{-c_H} \cdot S^{\widehat{\alpha}} \cdot \prod_{i=1}^{4} T_i^{\widehat{u}_i} \mod n$

Add to $\widehat{\mathcal{T}}$: $\{\widehat{T}_i\}, \widehat{T}_\Delta, \widehat{Q}$

Compute: $\widehat{c}_H = Hash(\widehat{\mathcal{T}}, \mathcal{C}, n_1)$

Compare: $\widehat{c}_H, c_H$

Table 6: Verification of required predicates

$$\Delta = \begin{cases} z_j - m_j & \text{if } \leq \\ z_j - m_j - 1 & \text{if } < \\ m_j - z_j & \text{if } \geq \\ m_j - z_j - 1 & \text{if } > \end{cases} \qquad a = \begin{cases} -1 & \text{if } (\leq \vee <) \\ 1 & \text{if } (\geq \vee >) \end{cases}$$

$$\Delta' = \begin{cases} z_j & \text{if } \leq \\ z_j - 1 & \text{if } < \\ z_j & \text{if } \geq \\ z_j + 1 & \text{if } > \end{cases}$$

The omitted calculation of the variables $\{\widehat{T_i}\}, \widehat{T_\Delta}, \widehat{Q}$ is presented in the following equations.

$$\begin{aligned} \widehat{T_i} &= T_i^{-c_H} \cdot Z^{\widehat{u_i}} \cdot S^{\widehat{r_i}} \\ &= (Z^{u_i} \cdot S^{r_i})^{-c_H} \cdot Z^{\widetilde{u_i} + c_H \cdot u_i} \cdot S^{\widetilde{r_i} + c_H \cdot r_i} \\ &= Z^{\widetilde{u_i}} \cdot S^{\widetilde{r_i}} \\ &= \overline{T}_i \end{aligned}$$

$$\begin{aligned} \widehat{T_\Delta} &= \left(T_\Delta^a \cdot Z^{\Delta'}\right)^{-c_H} \cdot Z^{\widehat{m_j}} \cdot S^{a \cdot \widehat{r}_\Delta} \\ &= \left((Z^\Delta \cdot S^{r_\Delta})^a \cdot Z^{\Delta'}\right)^{-c_H} \cdot Z^{\widetilde{m_j} + c_H \cdot m_j} \cdot S^{a \cdot (\widetilde{r}_\Delta + c_H \cdot r_\Delta)} \\ &= \left(Z^{\Delta \cdot a + \Delta'}\right)^{-c_H} \cdot Z^{\widetilde{m_j} + c_H \cdot m_j} \cdot S^{-a \cdot r_\Delta \cdot c_H} \cdot S^{a \cdot (\widetilde{r}_\Delta + c_H \cdot r_\Delta)} \\ &= \left(Z^{m_j}\right)^{-c_H} \cdot Z^{\widetilde{m_j} + c_H \cdot m_j} \cdot S^{a \cdot \widetilde{r}_\Delta} \\ &= Z^{\widetilde{m_j}} \cdot S^{a \cdot \widetilde{r}_\Delta} \\ &= \overline{T}_\Delta \end{aligned}$$

$$\begin{aligned} \widehat{Q} &= T_\Delta^{-c_H} \cdot S^{\widehat{\alpha}} \cdot \prod_{i=1}^{4} T_i^{\widehat{u}_i} \\ &= (Z^\Delta \cdot S^{r_\Delta})^{-c_H} \cdot S^{\widetilde{\alpha} + c_H \cdot (r_\Delta - u_1 r_1 - u_2 r_2 - u_3 r_3 - u_4 r_4)} \cdot \prod_{i=1}^{4} T_i^{\widetilde{u}_i + c_H \cdot u_i} \\ &= Z^{-c_H \cdot \Delta} \cdot S^{-c_H \cdot r_\Delta} \cdot S^{\widetilde{\alpha} + c_H \cdot r_\Delta} \cdot \prod_{i=1}^{4} (Z^{u_i} \cdot S^{r_i})^{\widetilde{u}_i + c_H \cdot u_i} \cdot S^{-c_H \cdot u_i r_i} \\ &= Z^{-c_H \cdot (u_1^2 + u_2^2 + u_3^2 + u_4^2)} \cdot S^{\widetilde{\alpha}} \cdot \prod_{i=1}^{4} Z^{u_i \cdot \widetilde{u}_i + c_H \cdot u_i^2} \cdot S^{r_i \cdot \widetilde{u}_i} \\ &= S^{\widetilde{\alpha}} \cdot \prod_{i=1}^{4} Z^{-c_H \cdot u_i^2} \cdot Z^{u_i \cdot \widetilde{u}_i + c_H \cdot u_i^2} \cdot S^{r_i \cdot \widetilde{u}_i} \\ &= S^{\widetilde{\alpha}} \cdot \prod_{i=1}^{4} T_i^{\widetilde{u}_i} \\ &= Q \end{aligned}$$

As seen in the equations above, the calculated values indeed match with the original ones if the tuple of sent values was correctly generated. In this case, the calculated hash c_H matches as well. Note that the verifier only gains information on the validity of the predicate set while the concrete attributes stay hidden.

6 Credential linking

In some situations, a proof of possession of multiple credentials issued to the same person is required. That requires multiple credentials for the same holder to be cryptographically linked.

To allow this, a long-term private ID is inserted into every credential as value m_1 during the issuing process. Every holder initially (during the holder's wallet setup) generates this private ID which is then used throughout all of his credentials.

Whenever the holder seeks to request a new credential, the holder sends a committed version of his private ID to the issuer. This can be achieved by calculating $R_1^{m_1}$ from m_1 and R_1, which is part of the issuer's public key. The rest of the credential issuing protocol stays the same (see Table 2).

In the case of the verifier requesting the verification of multiple credentials $\{\mathcal{C}_p\}_i$, the protocol is augmented with an additional proof. The holder does not only need to proof the authenticity of each credential $\mathcal{C}_p$ and the validity of the corresponding attribute predicates $\mathcal{D}_i$, but the holder also needs to show that all credentials are linked to the same holder. This is done by setting the random value $\widetilde{m}_1$ in protocol 5.2 to the same value for all credentials.

This mechanism shows the equality of attribute m_1 for all presented credentials and thereby proves that the credentials were issued to the same person.

7 Credential revocation

Issuers need to be capable to revoke credentials issued by them. Holders need to have the ability to proof to a verifier that their credential is valid, i.e. has not been revoked without leaking any other information.

The basic idea is to have the issuer publish an accumulator on the public ledger. At all times, this accumulator represents the product of values representing the currently non-revoked credentials. A holder seeking to prove his non-revocation status in fact proves that the factor corresponding to his credential is still part of the overall product. Issuing a new credential or, alternatively, revoking an existing credential, can then be realized by simply adding a new factor or removing an existing factor from the accumulator, respectively.

In AnonCreds 1.0, revocation with type-3 pairings is implemented based on work by Camenisch et al. [CKS08].

7.1 Non-revocation credential issuing

Revocation is realized by augmenting the credential $\mathcal{C}_p$ with an additional non-revocation credential $\mathcal{C}_{NR}$.

7.1.1 Abstract overview

For a given credential $\mathcal{C}_p$, the issuer computes a corresponding non-revocation credential $\mathcal{C}_{NR}$. Afterwards, the revocation information (V, acc_V) on the public ledger is updated.

Holder		Issuer		Public Ledger
		Compute: $\mathcal{C}_{NR}$ from $\mathcal{C}_p$		
	$\xleftarrow{\mathcal{C}_{NR}}$			
Store: $\mathcal{C}_{NR}$ alongside $\mathcal{C}_p$				
		Compute: new (V, acc_V)		
			$\xrightarrow{\text{update } (V, acc_V)}$	
				Store: received values

Table 7: Abstract revocation issuing protocol

7.1.2 Issuer preparation

Before issuing non-revocation credentials, some initial setup by the issuer is necessary. The first operation during the issuer preparation is determining a pairing operation scheme. For some choice of multiplicative groups $\mathcal{G}_1 = \langle g \rangle, \mathcal{G}_2 = \langle g' \rangle, \mathcal{G}_T$ of order q (for some prime q), the issuer selects a type-3 pairing operation $e : \mathcal{G}_1 \times \mathcal{G}_2 \to \mathcal{G}_T$.

Issuer		Public Ledger
Set up: pairing operation scheme		
Sample: $h, h_0, h_1, h_2, \widetilde{h} \in \mathbb{G}_1$ $u, \widehat{h} \in \mathbb{G}_2$ $sk, x \in \mathbb{F}_q$		
Compute: $pk = g^{sk}$ $y = \widehat{h}^x$		
Sample: $\gamma \in \mathbb{F}_q$		
Compute: $g_1, g_2 \dots, g_L, g_{L+2}, \dots, g_{2L}$ where $g_i = g^{\gamma^i}$ $g'_1, g'_2 \dots, g'_L, g'_{L+2}, \dots, g'_{2L}$ where $g'_i = g'^{\gamma^i}$ $z = e(g, g')^{\gamma^{L+1}}$		
Set: $V = \emptyset, acc_V = 1$		
Store: sk, x, γ		
	$\xrightarrow{h, h_0, h_1, h_2, \widetilde{h}, \widehat{h}, u, pk, y, z, V, acc_V}$	
		Store: received values

Table 8: Issuer revocation preparation

7.1.3 Issuing of non-revocation credentials

The goal of this protocol is to create a non-revocation credential $\mathcal{C}_{NR}$ for a given credential $\mathcal{C}_p$ with revocation attribute m_2. To achieve this, the issuer uses an accumulator (V, acc_V) with $V \subseteq \{1, \dots, L\}$. At any time, V is the set of all indices corresponding to non-revoked credentials.

Holder		**Issuer**
Retrieve from Ledger: h_2		
Sample: $s' \in \mathbb{F}_q$		
Compute: $U_R = h_2^{s'}$		
	$\xrightarrow{U_R}$	
		Assign: index $i < L$ to holder
		Sample: $s'', c \in \mathbb{F}_q$
		Retrieve from Ledger: (V, acc_V)
		Compute: $\sigma = (h_0 \cdot h_1^{m_2} \cdot U_R \cdot g_i \cdot h_2^{s''})^{\frac{1}{x+c}}$ $\sigma_i = g'^{\frac{1}{sk+\gamma^i}}$ $w = \prod_{j \in V} g'_{L+1-j+i}$ $u_i = u^{\gamma^i}$
		Update on Ledger: $acc_V \leftarrow acc_V \cdot g'_{L+1-i}$ $V \leftarrow V \cup \{i\}$
	$\xleftarrow{(\sigma, c, s'', g_i, g'_i, i, \sigma_i, u_i, w, V)}$	
Compute: $s = s' + s''$		
Store: $\mathcal{C}_{NR} = (\sigma, c, s, g_i, g'_i, i, \sigma_i, u_i, w, V)$		

Table 9: Non-revocation credential issuing protocol

From this credential issuing protocol we can observe that the state of the accumulator is equal to

$$acc_V = \prod_{j \in V} g'_{L+1-j}$$

at all times. We can also observe a direct link between the public accumulator value acc_V and any individual holder and his identifier g_i through

$$\begin{aligned} e(g_i, acc_V) &= e(g^{\gamma \cdot i}, \prod_{j \in V} g'^{\gamma \cdot (L+1-j)}) \\ &= e(g, \prod_{j \in V} g'^{\gamma \cdot (L+1-j+i)}) \\ &= e(g, g')^{\gamma \cdot (L+1)} \cdot e(g, \prod_{j \in V \setminus \{i\}} g'_{L+1-j+i}) \\ &= z \cdot e(g, w) \end{aligned}$$

The value z is part of the accumulator public key. The remaining holder-specific variable w is a so-called 'witness', which acts as a sort of counter-value to g_i. The combination of these two values can then prove that the holder is included within the public accumulator.

7.2 Non-revocation credential verification

Before verifying the non-revocation status of a credential $\mathcal{C}_p$, both, the holder and the verifier, retrieve the current version of the public accumulator (V, acc_V) from the ledger. The holder then creates a non-revocation proof based on the updated values $\mathcal{C}_{NR}, V, acc_V$ and sends it to the verifier.

7.2.1 Abstract overview

Holder	**Verifier**	**Public Ledger**
	$\xleftarrow{\text{Retrieve current } (V, acc_V)}$	
Update: $\mathcal{C}_{NR}$ from $\mathcal{C}_p, (V, acc_V)$		
Create: proof $\mathcal{P}_{NR}$		
$\xrightarrow{\mathcal{P}_{NR}}$		
		$\xleftarrow{\text{Retrieve current } (V, acc_V)}$
	Verify: tuple $(\mathcal{P}, \mathcal{P}_{NR})$	

Table 10: Abstract non-revocation verification protocol

7.2.2 Verification of non-revocation credentials

To build a verification proof that a holder's credential $\mathcal{C}_p$ has not been revoked yet, the holder prepares a non-revocation proof. To generate that proof, the holder uses the issuer's public revocation key $(h, h_0, h_1, h_2, \widetilde{h}, \widehat{h}, u, pk, y)$ as well as his own non-revocation credential $\mathcal{C}_{NR} = (\sigma, c, s, g_i, g'_i, i, \sigma_i, u_i, w_{old}, V_{old})$. Due to the length of this protocol, we decided not to present it in a sequence diagram as in the other sections.

Holder:

- Retrieve from Ledger: (V, acc_V)
- Compute $w = w_{old} \cdot \frac{\prod_{j \in V \setminus V_{old}} g'_{L+1-j+i}}{\prod_{j \in V_{old} \setminus V} g'_{L+1-j+i}}$
- Sample: $\rho, \rho', r, r', r'', r''', o, o' \in \mathbb{F}_q$
- Compute:

$$\begin{aligned} E &\leftarrow h^{\rho}\widetilde{h}^{o} & D &\leftarrow g^{r}\widetilde{h}^{o'}; \\ A &\leftarrow \sigma\widetilde{h}^{\rho} & \mathcal{G} &\leftarrow g_i\widetilde{h}^{r}; \\ \mathcal{W} &\leftarrow w\widehat{h}^{r'} & \mathcal{S} &\leftarrow \sigma_i\widehat{h}^{r''} \\ \mathcal{U} &\leftarrow u_i\widehat{h}^{r'''} \end{aligned}$$

$$\begin{aligned} m &\leftarrow \rho \cdot c; & t &\leftarrow o \cdot c; \\ m' &\leftarrow r \cdot r''; & t' &\leftarrow o' \cdot r''; \end{aligned}$$

  and add all values to $\mathcal{C}$ (as defined in Section 5.2).

- Sample: $\widetilde{\rho}, \widetilde{o}, \widetilde{o'}, \widetilde{c}, \widetilde{m}, \widetilde{m'}, \widetilde{t}, \widetilde{t'}, \widetilde{m_2}, \widetilde{s}, \widetilde{r}, \widetilde{r'}, \widetilde{r''}, \widetilde{r'''}, \in \mathbb{F}_q$

- Compute:

$$\begin{aligned}
&\overline{T_1} \leftarrow h^{\widetilde{\rho}}\widetilde{h}^{\widetilde{o}} && \overline{T_2} \leftarrow E^{\widetilde{c}}h^{-\widetilde{m}}\widetilde{h}^{-\widetilde{t}} \\
&\overline{T_4} \leftarrow e(\widetilde{h}, \mathrm{acc}_{\mathrm{V}})^{\widetilde{r}} \cdot e(1/g, \widehat{h})^{\widetilde{r'}} && \overline{T_5} \leftarrow g^{\widetilde{r}}\widetilde{h}^{\widetilde{o'}} \\
&\overline{T_6} \leftarrow D^{\widetilde{r''}}g^{-\widetilde{m'}}\widetilde{h}^{-\widetilde{t'}} && \overline{T_7} \leftarrow e(pk \cdot \mathcal{G}, \widehat{h})^{\widetilde{r''}} \cdot e(\widetilde{h}, \widehat{h})^{-\widetilde{m'}} \cdot e(\widetilde{h}, \mathcal{S})^{\widetilde{r}} \\
&\overline{T_8} \leftarrow e(\widetilde{h}, u)^{\widetilde{r}} \cdot e(1/g, \widehat{h})^{\widetilde{r'''}}
\end{aligned}$$

$$\overline{T_3} \leftarrow e(A, \widehat{h})^{\widetilde{c}} \cdot e(\widetilde{h}, \widehat{h})^{\widetilde{r}} \cdot e(\widetilde{h}, y)^{-\widetilde{\rho}} \cdot e(\widetilde{h}, \widehat{h})^{-\widetilde{m}} \cdot e(h_1, \widehat{h})^{-\widetilde{m_2}} \cdot e(h_2, \widehat{h})^{-\widetilde{s}}$$

  and add all values to $\mathcal{T}$ (as defined in Section 5.2).

- Note that the calculation $c_H = Hash(\mathcal{T}, \mathcal{C}, n_1)$ (as seen in Section 5.2) needs to reflect these additional values as well.

- Compute: $\forall x \in X := \{\rho, c, m, m', m_2, t, t', r, r', r'', r''', s, o, o'\}$

$$\hat{x} = \tilde{x} - c_H \cdot x \qquad\qquad \text{(i.e. } \hat{\rho} = \tilde{\rho} - c_H \cdot \rho \text{ and so on)}$$

  and add them to the initially empty set $\mathcal{X}$

- Send $(c_H, \mathcal{X}, \mathcal{C})$ to the verifier

Verifier:

- Compute:

$$\begin{aligned}
&\widehat{T_1} \leftarrow E^{c_H} \cdot h^{\widehat{\rho}} \cdot \widetilde{h}^{\widehat{o}} && \widehat{T_2} \leftarrow E^{\widehat{c}} \cdot h^{-\widehat{m}} \cdot \widetilde{h}^{-\widehat{t}} \\
&\widehat{T_4} \leftarrow \left(\frac{e(\mathcal{G}, \mathrm{acc}_{\mathrm{V}})}{e(g, \mathcal{W})z}\right)^{c_H} \cdot e(\widetilde{h}, \mathrm{acc}_{\mathrm{V}})^{\widehat{r}} \cdot e(1/g, \widehat{h})^{\widehat{r'}} && \widehat{T_5} \leftarrow D^{c_H} \cdot g^{\widehat{r}}\widetilde{h}^{\widehat{o'}} \\
&\widehat{T_6} \leftarrow D^{\widehat{r''}} \cdot g^{-\widehat{m'}}\widetilde{h}^{-\widehat{t'}} && \widehat{T_7} \leftarrow \left(\frac{e(pk \cdot \mathcal{G}, \mathcal{S})}{e(g, g')}\right)^{c_H} \cdot e(pk \cdot \mathcal{G}, \widehat{h})^{\widehat{r''}} \cdot e(\widetilde{h}, \widehat{h})^{-\widehat{m'}} \cdot e(\widetilde{h}, \mathcal{S})^{\widehat{r}} \\
&\widehat{T_8} \leftarrow \left(\frac{e(\mathcal{G}, u)}{e(g, \mathcal{U})}\right)^{c_H} \cdot e(\widetilde{h}, u)^{\widehat{r}} \cdot e(1/g, \widehat{h})^{\widehat{r'''}}
\end{aligned}$$

$$\widehat{T_3} \leftarrow \left(\frac{e(h_0\mathcal{G}, \widehat{h})}{e(A, y)}\right)^{c_H} \cdot e(A, \widehat{h})^{\widehat{c}} \cdot e(\widetilde{h}, \widehat{h})^{\widehat{r}} \cdot e(\widetilde{h}, y)^{-\widehat{\rho}} \cdot e(\widetilde{h}, \widehat{h})^{-\widehat{m}} \cdot e(h_1, \widehat{h})^{-\widehat{m_2}} \cdot e(h_2, \widehat{h})^{-\widehat{s}}$$

  and add these values to $\widehat{\mathcal{T}}$.

- Note that the final hash in protocol Table 6 only matches if $\overline{T}_i = \widehat{T}_i$ for $1 \le i \le 8$.

7.2.3 Correctness of non-revocation verification

In the following equations we have verified that a correctly generated setup (based on a non-revoked credential $\mathcal{C}_p$) leads to $\widehat{T}_i = \overline{T}_i$ and therefore yields the verifier's acceptance of the underlying $\mathcal{C}_p$.

$$\begin{aligned}
\widehat{T_1} &= E^{c_H} \cdot h^{\widehat{\rho}} \cdot \widetilde{h}^{\widehat{o}} \\
&= (h^{\rho} \cdot \widetilde{h}^{o})^{c_H} \cdot h^{\widetilde{\rho} - c_H \cdot \rho} \cdot \widetilde{h}^{\widetilde{o} - c_H \cdot o} \\
&= h^{\widetilde{\rho}} \cdot \widetilde{h}^{\widetilde{o}} \\
&= \overline{T}_1
\end{aligned}$$

$$\begin{aligned}
\widehat{T}_2 &= E^{\widehat{c}} \cdot h^{-\widehat{m}} \cdot \widetilde{h}^{-\widehat{t}} \\
&= E^{\widetilde{c} - c_H \cdot c} \cdot h^{-\widetilde{m} + c_H \cdot m} \cdot \widetilde{h}^{-\widetilde{t} + c_H \cdot t} \\
&= E^{\widetilde{c}} \cdot (h^{\rho} \cdot \widetilde{h}^{o})^{-c_H \cdot c} \cdot h^{-\widetilde{m} + c_H \cdot \rho \cdot c} \cdot \widetilde{h}^{-\widetilde{t} + c_H \cdot o \cdot c} \\
&= E^{\widetilde{c}} \cdot h^{-\widetilde{m}} \cdot \widetilde{h}^{-\widetilde{t}} \\
&= \overline{T}_2
\end{aligned}$$

$$\begin{aligned}
\widehat{T}_3 &= \left(\frac{e(h_0 \cdot \mathcal{G}, \widehat{h})}{e(A, y)} \right)^{c_H} \cdot e(A, \widehat{h})^{\widehat{c}} \cdot e(\widetilde{h}, \widehat{h})^{\widehat{r}} \cdot e(\widetilde{h}, y)^{-\widehat{\rho}} \cdot e(\widetilde{h}, \widehat{h})^{-\widehat{m}} \cdot e(h_1, \widehat{h})^{-\widehat{m}_2} \cdot e(h_2, \widehat{h})^{-\widehat{s}} \\
&= \left(\frac{e(h_0 \cdot \mathcal{G}, \widehat{h})}{e(A, y)} \right)^{c_H} \cdot e(A, \widehat{h})^{\widetilde{c} - c_H \cdot c} \cdot e(\widetilde{h}, \widehat{h})^{\widetilde{r} - c_H \cdot r} \cdot e(\widetilde{h}, y)^{-\widetilde{\rho} + c_H \cdot \rho} \cdot e(\widetilde{h}, \widehat{h})^{-\widetilde{m} + c_H \cdot m} \cdot e(h_1, \widehat{h})^{-\widetilde{m}_2 + c_H \cdot m_2} \cdot e(h_2, \widehat{h})^{-\widetilde{s} + c_H \cdot s} \\
&= e(A, \widehat{h})^{\widetilde{c}} \cdot e(\widetilde{h}, \widehat{h})^{\widetilde{r}} \cdot e(\widetilde{h}, y)^{-\widetilde{\rho}} \cdot e(\widetilde{h}, \widehat{h})^{-\widetilde{m}} \cdot e(h_1, \widehat{h})^{-\widetilde{m_2}} \cdot e(h_2, \widehat{h})^{-\widetilde{s}} \\
&\quad \cdot \left(\frac{e(h_0 \cdot \mathcal{G}, \widehat{h})}{e(A, y)} \cdot e(A, \widehat{h})^{-c} \cdot e(\widetilde{h}, \widehat{h})^{-r} \cdot e(\widetilde{h}, y)^{\rho} \cdot e(\widetilde{h}, \widehat{h})^{m} \cdot e(h_1, \widehat{h})^{m_2} \cdot e(h_2, \widehat{h})^{s} \right)^{c_H} \\
&= \overline{T}_3 \cdot \left(\frac{e(h_0 \cdot (g_i \cdot \widetilde{h}^r), \widehat{h})}{e(\sigma \cdot \widetilde{h}^{\rho}, \widehat{h}^x)} \cdot e(\sigma \cdot \widetilde{h}^{\rho}, \widehat{h})^{-c} \cdot e(\widetilde{h}, \widehat{h})^{-r} \cdot e(\widetilde{h}, \widehat{h}^x)^{\rho} \cdot e(\widetilde{h}, \widehat{h})^{m} \cdot e(h_1, \widehat{h})^{m_2} \cdot e(h_2, \widehat{h})^{s} \right)^{c_H} \\
&= \overline{T}_3 \cdot \left(\frac{e(h_0 \cdot g_i \cdot \widetilde{h}^r, \widehat{h})}{e(\sigma \cdot \widetilde{h}^{\rho}, \widehat{h}^x)} \cdot e(\sigma, \widehat{h})^{-c} \cdot e(\widetilde{h}, \widehat{h})^{-c \cdot \rho} \cdot e(\widetilde{h}, \widehat{h})^{-r} \cdot e(\widetilde{h}, \widehat{h})^{\rho \cdot x} \cdot e(\widetilde{h}, \widehat{h})^{\rho \cdot c} \cdot e(h_1, \widehat{h})^{m_2} \cdot e(h_2, \widehat{h})^{s} \right)^{c_H} \\
&= \overline{T}_3 \cdot \left(\frac{e(h_0 \cdot g_i \cdot \widetilde{h}^r, \widehat{h}) \cdot e(\widetilde{h}, \widehat{h})^{\rho \cdot x} \cdot e(h_1, \widehat{h})^{m_2} \cdot e(h_2, \widehat{h})^{s}}{e(\sigma \cdot \widetilde{h}^{\rho}, \widehat{h}^x) \cdot e(\sigma, \widehat{h})^{c} \cdot e(\widetilde{h}, \widehat{h})^{r}} \right)^{c_H} \\
&= \overline{T}_3 \cdot \left(\frac{e(h_0, \widehat{h}) \cdot e(g_i, \widehat{h}) \cdot e(\widetilde{h}, \widehat{h})^{r} \cdot e(\widetilde{h}, \widehat{h})^{\rho \cdot x} \cdot e(h_1, \widehat{h})^{m_2} \cdot e(h_2, \widehat{h})^{s}}{e(\sigma, \widehat{h})^{x} \cdot e(\widetilde{h}, \widehat{h})^{\rho \cdot x} \cdot e(\sigma, \widehat{h})^{c} \cdot e(\widetilde{h}, \widehat{h})^{r}} \right)^{c_H} \\
&= \overline{T}_3 \cdot \left(\frac{e(h_0, \widehat{h}) \cdot e(g_i, \widehat{h}) \cdot e(\widetilde{h}, \widehat{h})^{r} \cdot e(h_1, \widehat{h})^{m_2} \cdot e(h_2, \widehat{h})^{s}}{e(h_0 \cdot h_1^{m_2} \cdot h_2^{s'} \cdot g_i \cdot h_2^{s''}, \widehat{h})^{\frac{x+c}{x+c}} \cdot e(\widetilde{h}, \widehat{h})^{r}} \right)^{c_H} \\
&= \overline{T}_3 \cdot \left(\frac{e(h_0, \widehat{h}) \cdot e(g_i, \widehat{h}) \cdot e(\widetilde{h}, \widehat{h})^{r} \cdot e(h_1, \widehat{h})^{m_2} \cdot e(h_2, \widehat{h})^{s}}{e(h_0, \widehat{h}) \cdot e(h_1, \widehat{h})^{m_2} \cdot e(h_2, \widehat{h})^{s' + s''} \cdot e(g_i, \widehat{h}) \cdot e(\widetilde{h}, \widehat{h})^{r}} \right)^{c_H} \\
&= \overline{T}_3
\end{aligned}$$

$$\begin{aligned}
\widehat{T}_4 &= \left(\frac{e(\mathcal{G}, acc_V)}{e(g, \mathcal{W}) \cdot z} \right)^{c_H} \cdot e(\widetilde{h}, acc_V)^{\widehat{r}} \cdot e(g^{-1}, \widehat{h})^{\widehat{r'}} \\
&= e(\widetilde{h}, acc_V)^{\widetilde{r}} \cdot e(g^{-1}, \widehat{h})^{\widetilde{r'}} \cdot \left(\frac{e(\mathcal{G}, acc_V)}{e(g, \mathcal{W}) \cdot z} \cdot e(\widetilde{h}, acc_V)^{-r} \cdot e(g^{-1}, \widehat{h})^{-r'} \right)^{c_H} \\
&= \overline{T}_4 \cdot \left(\frac{e(g_i \cdot \widetilde{h}^r, acc_V)}{e(g, w \cdot \widehat{h}^{r'}) \cdot z} \cdot e(\widetilde{h}, acc_V)^{-r} \cdot e(g, \widehat{h})^{r'} \right)^{c_H} \\
&= \overline{T}_4 \cdot \left(\frac{e(g_i, acc_V) \cdot e(\widetilde{h}, acc_V)^{r}}{e(g, w) \cdot e(g, \widehat{h})^{r'} \cdot z} \cdot e(\widetilde{h}, acc_V)^{-r} \cdot e(g, \widehat{h})^{r'} \right)^{c_H} \\
&= \overline{T}_4 \cdot \left(\frac{e(g_i, acc_V)}{e(g, w) \cdot z} \right)^{c_H} \\
&= \overline{T}_4 \cdot \left(\frac{e(g_i, \prod_{j \in V} g'_{L+1-j})}{e(g, \prod_{j \in V}^{j \neq i} g'_{L+1-j+i}) \cdot e(g, g')^{\gamma^{L+1}}} \right)^{c_H} \\
&= \overline{T}_4 \cdot \left(\frac{e(g, g')^{\sum_{j \in V} \gamma^{L+1-j+i}}}{e(g, g')^{\sum_{j \in V} \gamma^{L+1-j+i}}} \right)^{c_H} \\
&= \overline{T}_4
\end{aligned}$$

$$
\begin{aligned}
\widehat{T}_5 &= \mathcal{D}^{c_H} \cdot g^{\widehat{r}} \cdot \widetilde{h}^{\widehat{o'}} \\
&= g^{\widetilde{r}} \cdot \widetilde{h}^{\widetilde{o'}} \cdot (\mathcal{D} \cdot g^{-r} \cdot \widetilde{h}^{-o'})^{c_H} \\
&= \overline{T_5} \cdot (g^{r} \cdot \widetilde{h}^{o'} \cdot g^{-r} \cdot \widetilde{h}^{-o'})^{c_H} \\
&= \overline{T}_5
\end{aligned}
$$

$$
\begin{aligned}
\widehat{T}_6 &= D^{\widehat{r}''} \cdot g^{-\widehat{m}'} \cdot \widetilde{h}^{-\widehat{t}'} \\
&= D^{\widetilde{r}''} \cdot g^{-\widetilde{m}'} \cdot \widetilde{h}^{-\widetilde{t}'} \cdot (D^{-r''} \cdot g^{m'} \cdot \widetilde{h}^{t'})^{c_H} \\
&= \overline{T}_6 \cdot (g^{-r'' \cdot r} \cdot \widetilde{h}^{-r'' \cdot o'} \cdot g^{r \cdot r''} \cdot \widetilde{h}^{o' \cdot r''})^{c_H} \\
&= \overline{T}_6
\end{aligned}
$$

$$
\begin{aligned}
\widehat{T}_7 &= \left(\frac{e(pk \cdot \mathcal{G}, \mathcal{S})}{e(g, g')}\right)^{c_H} \cdot e(pk \cdot \mathcal{G}, \widehat{h})^{\widehat{r}''} \cdot e(\widetilde{h}, \widehat{h})^{-\widehat{m}'} \cdot e(\widetilde{h}, \mathcal{S})^{\widehat{r}} \\
&= \left(\frac{e(pk \cdot \mathcal{G}, \mathcal{S})}{e(g, g')}\right)^{c_H} \cdot e(pk \cdot \mathcal{G}, \widehat{h})^{\widetilde{r}'' - c_H \cdot r''} \cdot e(\widetilde{h}, \widehat{h})^{-\widetilde{m}' + c_H \cdot m'} \cdot e(\widetilde{h}, \mathcal{S})^{\widetilde{r} - c_H \cdot r} \\
&= e(pk \cdot \mathcal{G}, \widehat{h})^{\widetilde{r}''} \cdot e(\widetilde{h}, \widehat{h})^{-\widetilde{m}'} \cdot e(\widetilde{h}, \mathcal{S})^{\widetilde{r}} \cdot \left(\frac{e(pk \cdot \mathcal{G}, \mathcal{S})}{e(g, g')} \cdot e(pk \cdot \mathcal{G}, \widehat{h})^{-r''} \cdot e(\widetilde{h}, \widehat{h})^{m'} \cdot e(\widetilde{h}, \mathcal{S})^{-r}\right)^{c_H} \\
&= \overline{T_7} \cdot \left(\frac{e(pk \cdot \mathcal{G}, \sigma_i \cdot \widehat{h}^{r''})}{e(g, g')} \cdot e(pk \cdot \mathcal{G}, \widehat{h})^{-r''} \cdot e(\widetilde{h}, \widehat{h})^{r \cdot r''} \cdot e(\widetilde{h}, \sigma_i \cdot \widehat{h}^{r''})^{-r}\right)^{c_H} \\
&= \overline{T_7} \cdot \left(\frac{e(pk \cdot \mathcal{G}, \sigma_i) \cdot e(pk \cdot \mathcal{G}, \widehat{h})^{r''}}{e(g, g')} \cdot e(pk \cdot \mathcal{G}, \widehat{h})^{-r''} \cdot e(\widetilde{h}, \widehat{h})^{r \cdot r''} \cdot e(\widetilde{h}, \sigma_i)^{-r} \cdot e(\widetilde{h}, \widehat{h})^{-r \cdot r''}\right)^{c_H} \\
&= \overline{T_7} \cdot \left(\frac{e(pk \cdot \mathcal{G}, \sigma_i)}{e(g, g')} \cdot e(\widetilde{h}, \sigma_i)^{-r}\right)^{c_H} \\
&= \overline{T_7} \cdot \left(\frac{e(g^{sk} \cdot g_i \cdot \widetilde{h}^{r}, g'^{\frac{1}{sk + \gamma^i}})}{e(g, g')} \cdot e(\widetilde{h}, g'^{\frac{1}{sk + \gamma^i}})^{-r}\right)^{c_H} \\
&= \overline{T_7} \cdot \left(\frac{e(g, g')^{\frac{sk}{sk + \gamma^i}} \cdot e(g_i, g')^{\frac{1}{sk + \gamma^i}} \cdot e(\widetilde{h}, g')^{\frac{r}{sk + \gamma^i}}}{e(g, g')} \cdot e(\widetilde{h}, g')^{\frac{-r}{sk + \gamma^i}}\right)^{c_H} \\
&= \overline{T_7} \cdot \left(\frac{e(g, g')^{\frac{sk}{sk + \gamma^i}} \cdot e(g_i, g')^{\frac{1}{sk + \gamma^i}}}{e(g, g')}\right)^{c_H} \\
&= \overline{T_7} \cdot \left(\frac{e(g, g')^{\frac{sk}{sk + \gamma^i}} \cdot e(g, g')^{\frac{\gamma^i}{sk + \gamma^i}}}{e(g, g')}\right)^{c_H} \\
&= \overline{T}_7
\end{aligned}
$$

$$\begin{aligned}
\widehat{T}_8 &= \left(\frac{e(\mathcal{G},u)}{e(g,\mathcal{U})}\right)^{c_H} \cdot e(\widetilde{h},u)^{\widehat{r}} \cdot e(g^{-1},\widehat{h})^{\widehat{r}'''} \\
&= \left(\frac{e(g_i \cdot \widetilde{h}^{r},u)}{e(g,u_i \cdot \widehat{h}^{r'''})}\right)^{c_H} \cdot e(\widetilde{h},u)^{\widetilde{r}-c_H \cdot r} \cdot e(g^{-1},\widehat{h})^{\widetilde{r}'''-c_H \cdot r'''} \\
&= e(\widetilde{h},u)^{\widetilde{r}} \cdot e(g^{-1},\widehat{h})^{\widetilde{r}'''} \cdot \left(\frac{e(g_i \cdot \widetilde{h}^{r},u)}{e(g,u_i \cdot \widehat{h}^{r'''})} \cdot e(\widetilde{h},u)^{-r} \cdot e(g^{-1},\widehat{h})^{-r'''}\right)^{c_H} \\
&= e(\widetilde{h},u)^{\widetilde{r}} \cdot e(g^{-1},\widehat{h})^{\widetilde{r}'''} \cdot \left(\frac{e(g_i,u) \cdot e(\widetilde{h},u)^{r}}{e(g,u_i) \cdot e(g,\widehat{h})^{r'''}} \cdot e(\widetilde{h},u)^{-r} \cdot e(g^{-1},\widehat{h})^{-r'''}\right)^{c_H} \\
&= e(\widetilde{h},u)^{\widetilde{r}} \cdot e(g^{-1},\widehat{h})^{\widetilde{r}'''} \cdot \left(\frac{e(g_i,u)}{e(g,u_i)}\right)^{c_H} \\
&= e(\widetilde{h},u)^{\widetilde{r}} \cdot e(g^{-1},\widehat{h})^{\widetilde{r}'''} \cdot \left(\frac{e(g,u)^{\gamma \cdot i}}{e(g,u)^{\gamma \cdot i}}\right)^{c_H} \\
&= e(\widetilde{h},u)^{\widetilde{r}} \cdot e(g^{-1},\widehat{h})^{\widetilde{r}'''} \\
&= \overline{T}_8
\end{aligned}$$

7.3 Revoking credentials

When revoking a holder's credential $\mathcal{C}_p$, the issuer retrieves the corresponding index i from his local database. The index is then removed from the currently valid index set V and the accumulator value is updated accordingly, i.e. the factor corresponding to said credential is removed.

Issuer		**Public Ledger**
Retrieve: holder's index i		
Update: $V \leftarrow V \setminus \{i\}$ $acc_V \leftarrow \frac{acc_V}{g'_{L+1-i}}$		
	$\xrightarrow{(V,acc_V)}$	
		Update: received values

Table 11: Credential revocation

Part III
Theoretical Analysis

8 CL Signatures

In this section, a theoretical analysis of the usage of CL signatures within Hyperledger's AnonCreds 1.0 protocol is given. In particular, we will focus on deviations between the theoretical work done by Camenisch[CG08] and the specification of AnonCreds 1.0 [KL05].

8.1 On CL signatures in general

CL signatures were shown to be secure under the strong RSA assumption as presented in [CG08]. The strong RSA assumption states the following:

Assumption 1. (Strong RSA Assumption)
Given RSA modulus n and an element $u \in \mathbb{Z}_n^*$ it is hard to compute values $e > 1$ and v such that $v^e \equiv u \; mod \; n$.

Informally, the assumption claims that the RSA problem is hard even if an adversary is able to choose the public exponent e.

Earlier on, the strong RSA assumption provided a basis for practical and provably secure signature schemes under the standard assumption without using the random oracle model. In 1999, Gennaro et al. and Cramer et al. respectively proposed such signature schemes [GHR99][CS99]. Since then, different approaches based on the strong RSA assumption followed, for example, the Fischlin scheme in 2003 [Fis02], a scheme by Tan in 2006 [Tan06] and the CL signature scheme in 2002, which we consider in this paper.

CL signatures already find use in several contexts. In smart grid systems consumption data is usually disclosed to utility providers for time-of-use billing, profiling or forecasting. In [RD11], a CL signature based protocol was introduced in order to allow consumers to proof correctness of consumption calculations without revealing any consumption data.

Other applications are off-line anonymous e-cash schemes [CHL05], which allow users to withdraw coins and spend them in a non-linkable way.

Moreover, CL signatures play a leading part in TCG (Trusted Computing Group) attestation. For example, Direct Anonymous Attestation (DAA) uses CL signatures to enable remote authentication of a trusted computer whilst preserving the privacy of the platform's user. DAA has been adopted by the TCG in version 1.2 of its Trusted Platform Module specification [BCC04]. Beside this, there exists a property-based approach for attestation containing CL signatures [CLL+06].

8.2 CL Key Generation in AnonCreds

As seen in Section 2, the CL key generation algorithm requires the sampling of $L+2$ quadratic residues, namely $S, Z, R_1, \ldots, R_L$. In theory, these should be randomly sampled from QR_n, i.e. the set of all quadratic residues modulo n.

AnonCreds 1.0 randomly samples the value S according to theory. Then, it generates the values $Z, R_1, \ldots, R_L$ by sampling random powers of S, thereby possibly deviating from the theoretical requirement.

To be more exact, $L+1$ random integer values $x_1, \ldots, x_{L+1}$ in the range $[2, p'q']$ with $n = (2p'+1) \cdot (2q'+1)$ are sampled. These are then used to generate the values $Z, R_1, \ldots, R_L$ as $S^{x_1}, \ldots, S^{x_{L+1}}$.

While it is clear that the resulting values are indeed elements of QR_n, it is not obvious why this approach would ensure yielding uniformly random or close to uniformly random samples from QR_n.

However, we claim that this algorithmic approach is practically as secure as the theoretical one. We show, that the probability for $S^x = S^y \bmod n$ for randomly chosen $x, y \in \{2, \ldots, p'q'\}$ is negligible.

For this, let $x, y \in \{2, \ldots, p'q'\}$ with $S^x = S^y \bmod n$. Then, $S^x/S^y = S^{x-y} = 1 \bmod n$, hence ord(S) $\mid x - y$, so ord(S) $\leq x - y$. As ord(S) $\in \{p'q', 2p'q'\}$ with probability $1 - 1/\text{poly}(n)$, it follows $x - y \geq p'q'$ in this case, a contradiction to $x, y \in \{2, \ldots, p'q'\}$.

Let us estimate the size of the set QR_n. By definition, elements of QR_n are from $\mathbb{Z}_n^*$ which has a size of $\phi(n) = (p-1) \cdot (q-1) = 2p' \cdot 2q' = 4p'q'$. Due to the composite nature of $n = p \cdot q$, the Chinese remainder theorem shows us that every quadratic residue possesses 4 distinct square roots. Therefore, the size of QR_n is exactly $4p'q'/4 = p'q'$.

Combining these two observations, we can see that for almost all choices of S the construction above indeed generates $p'q' - 1$ distinct quadratic residues, i.e. the entire set QR_n (only skipping $S^1 = S$). This indeed results in a uniformly random sampling of quadratic residues.

9 Credential Issuing and Verification

The credential issuing and verification mechanisms of AnonCreds 1.0 are based on Idemix [Res10] and are identical in their basic functioning. Idemix is a convolution of multiple secure cryptographic protocols with the goal of enhancing privacy within public-key infrastructures in order to build an anonymous credential system.

While there does not exist a rigorous security proof with a mathematical reduction from the entire Idemix system to CL signatures (and therefore the strong RSA assumption), the system is believed to be practically secure [Res10].

In Table 2, Table 5 and Table 6 we have given a detailed overview of the mathematical credential issuing process, credential verification process and predicate verification process, respectively. Further, at the end of Section 5, we have mathematically verified that a verifier does accept a given proof if the tuple of values sent by a holder was indeed correctly generated and is based on a valid CL signature. We have also verified in Section 5.2 that a predicate proof will be accepted by a verifier if the corresponding credential actually fulfills the attribute requirements. The cryptographic operations within these protocols seem strong and, for us, forging a proof seems to be infeasible as the CL signature functions as trust anchor for such claims.

10 Credential Revocation

AnonCreds 1.0 uses a type-3 variant of a secure type-1 pairing revocation scheme [CKS08]. Even though there is no security proof for this variant, it preserves the same basic structure and looks at least as secure as the original scheme, which is provably secure.

During the revocation verification, a crucial step is to verify that a given credential $\mathcal{C}_p$ is indeed linked to a corresponding non-revocation credential $\mathcal{C}_{NR}$. If left unchecked, this opens up possible attacks where a revoked credential is combined with another holder's valid non-revocation credential leading to the acceptance of the revoked credential.

In our opinion, the revocation part needs to contain some kind of proof that the value m_2 used for the calculation of the non-revocation proof is indeed contained in $\mathcal{C}_p$. A blinded version $\widehat{m_2} = \widetilde{m_2} - c_H \cdot m_2$ is sent to the verifier twice as part of the credential verification and the revocation proof. We believe that this needs to be the same value in order for the verifier to link both credentials $\mathcal{C}_p$ and $\mathcal{C}_{NR}$. However, the specification [KL05] suggests that the value $\widetilde{m}_2$ is sampled twice (page 7, 7.2, step 1 & page 7, non-revocation proof, step 8) resulting in different values $\widehat{m}$ in each of the two proofs.

We therefore believe that the verifier cannot check that a given non-revocation credential actually belongs to the given credential, thereby rendering him unable to securely check the revocation status of credentials at all.

This issue would be easily fixable by sampling $\widetilde{m_2}$ only once and using this value in both proofs. A review of the source-code (Ursa 0.3.2 and latest Ursa 0.3.7) strongly suggests that this issue is already fixed according to our suggestion above.

References

[BCC04] Ernest Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. https://eprint.iacr.org/2004/205.pdf, 01 2004.

[CG08] Jan Camenisch and Thomas Gross. Efficient attributes for anonymous credentials. https://eprint.iacr.org/2010/496.pdf, 01 2008.

[CHL05] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 302–321. Springer, 2005. https://eprint.iacr.org/2005/060.pdf.

[CKS08] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. https://eprint.iacr.org/2008/539.pdf, 2008.

[CL02] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. http://cs.brown.edu/people/alysyans/papers/camlys02b.pdf, 01 2002.

[CLL+06] Liqun Chen, Rainer Landfermann, Hans Löhr, Markus Rohe, Ahmad-Reza Sadeghi, and Christian Stüble. A protocol for property-based attestation. https://dl.acm.org/doi/10.1145/1179474.1179479, 2006.

[CS99] Ronald Cramer and Victor Shoup. Signature schemes based on the strong rsa assumption. Cryptology ePrint Archive, Report 1999/001, 1999. https://ia.cr/1999/001.

[Fis02] Marc Fischlin. The cramer-shoup strong-rsa signature scheme revisited, 2002. https://eprint.iacr.org/2002/017.pdf.

[GHR99] Rosario Gennaro, Shai Halevi, and Tal Rabin. Secure hash-and-sign signatures without the random oracle, 1999. https://eprint.iacr.org/1999/013.ps.

[KL05] Dmitry Khovratovich and Michael Lodder. Anonymous credentials with type-3 revocation. https://github.com/hyperledger/ursa-docs/tree/main/specs/anoncreds1, 06 2019, version 0.5.

[RD11] Alfredo Rial and George Danezis. Privacy-preserving smart metering, 2011. https://dl.acm.org/doi/10.1145/2046556.2046564.

[Res10] IBM Research. Specification of the identity mixer cryptographic library. https://dominoweb.draco.res.ibm.com/reports/rz3730_revised.pdf, 4 2010.

[Tan06] Chik How Tan. A secure signature scheme. https://dl.acm.org/doi/10.1145/1143549.1143589, 2006.

A Mathematical Foundations

Definition A.1. (Quadratic residue)
An element $a \in \mathbb{Z}_n^*$ is called quadratic residue, if there exists an element $b \in \mathbb{Z}_n^*$ with

$$b^2 = a \; mod \; n \; .$$

The set of all quadratic residues $\{a \in \mathbb{Z}_n^* \mid \exists b \in \mathbb{Z}_n^* : b^2 = a \; mod \; n\}$ is denoted as QR_n.

Definition A.2. (Safe prime)
A prime number p is called safe, if there exists a prime p' such that

$$p = 2p' + 1 \; .$$

The corresponding prime p' is also known as *Sophie Germain prime*.

Definition A.3. (Special RSA modulus)
An RSA modulus $n = pq$ is called special, if both p and q are safe primes.