Anonymous credentials with type 3 revocation

Dmitry Khovratovich Michael Lodder Cam Parra

25 April 2022 version 0.6

1 Introduction

1.1 Concept

The concept of *anonymous credentials* allows users to prove that their identity satisfies certain properties in an uncorrelated way without revealing other identity details. The properties can be raw identity attributes such as the birth date or the address, or more sophisticated predicates such as "A is older than 20 years old".

We assume three parties: *issuer*, *holder*, and *veri er*. From the functional perspective, the issuer gives a credential C based on identity schema X, which asserts certain properties $\mathcal{P}$ about X, to the holder. The credential consists of attributes represented by integers $m_1, m_2, \ldots, m_l$. The holder then presents $(\mathcal{P}, C)$ to the Verifier, which can verify that the issuer has asserted that holder's identity has property $\mathcal{P}$.

1.2 Properties

Credentials are *unforgeable* in the sense that no one can fool the Verifier with a credential not prepared by the issuer.

We say that credentials are *unlinkable* if it is impossible to correlate the presented credential across multiple presentations. This is implemented by the holder *proving* with a zero-knowledge proof *that he has a credential* rather than showing the credential.

Unlinkability can be simulated by the issuer generating a sufficient number of ordinary unrelated credentials. Also unlinkability can be turned off to make credentials *one-time use* so that second and later presentations are detected.

1.3 Pseudonyms

Typically a credential is bound to a certain pseudonym nym. It is supposed that holder has been registered as nym at the issuer, and communicated (part of) his identity X to him. After that the issuer can issue a credential that couples nym and X.

The holder may have a pseudonym at the Verifier, but not necessarily. If there is no pseudonym then the Verifier provides the service to users who did not register. If the pseudonym nym_V is required, it can be generated from a link secret m_1 together with nym in a way that nym can not be linked to nym_V. However, holder is supposed to prove that the credential presented was issued to a pseudonym derived from the same link secret as used to produce nym_V.

An identity owner also can create a policy address I that is used for managing agent proving authorization. The address are tied to credentials issued to holders such that agents cannot use these credentials without authorization.

2 Generic notation

Attribute m is a l_a-bit unsigned integer[1].

3 Protocol Overview

The described protocol supports anonymous credentials given to multiple holders by various issuers, which are presented to various relying parties.

Various types of anonymous credentials can be supported. In this section, the combination of CL-based credentials [**?**] and pairing-based revocation [**?**] is described.

The simplest credential lifecycle with one credential, single issuer, holder, and verifier is as follows:

[1] Technically it is possible to support credentials with di erent l , but in Sovrin for simplicity it is set l = 256.

1. Issuer determines a credential schema $\mathcal{S}$: the type of cryptographic signatures used to sign the credentials, the number l of attributes in a credential, the indices $A_h \subset \{1, 2, \ldots, l\}$ of hidden attributes, the public key P_k, the non-revocation credential attribute number l_r and non-revocation public key P_r (Section 4). Then he publishes it on the ledger and announces the attribute semantics.

2. Holder retrieves the credential schema from the ledger and sets the hidden attributes.

3. Holder requests a credential from issuer. He sends hidden attributes in a blinded form to issuer and agrees on the values of known attributes $A_k = \{1, 2, \ldots, l\} \setminus A_h$.

4. Issuer returns a credential pair (C_p, C_{NR}) to holder. The first credential contains the requested l attributes. The second credential asserts the non-revocation status of the first one. Issuer publishes the non-revoked status of the credential on the ledger.

5. Holder approaches verifier. Verifier sends the Proof Request $\mathcal{E}$ to holder. The Proof Request contains the credential schema $\mathcal{S}_E$ and disclosure predicates $\mathcal{D}$. The predicates for attribute m and value V can be of form $m = V$, $m < V$, or $m > V$. Some attributes may be asserted to be the same: $m_i = m_j$.

6. Holder checks that the credential pair he holds satisfy the schema $\mathcal{S}_E$. He retrieves the non-revocation witness from the ledger.

7. Holder creates a proof P that he has a non-revoked credential satisfying the proof request $\mathcal{E}$ and sends it to verifier.

8. Verifier verifies the proof.

If there are multiple issuers, the holder obtains credentials from them independently. To allow credential chaining, issuers reserve one attribute (usually m_1) for a secret value hidden by holder. Holder is supposed then to set it to the same value in all credentials, whereas Relying Parties require them to be equal along all credentials. A proof request should specify then a list of schemas that credentials should satisfy in certain order.

4 Schema preparation

Credentials should have limited use to only authorized holder entities called agents. Agents can prove authorization to use a credential by including a policy address I in primary credentials as attribute m_3.

4.1 Attributes

Issuer defines the primary credential schema $\mathcal{S}$ with l attributes $m_1, m_2, \ldots, m_l$ and the set of hidden attributes $A_h \subset \{1, 2, \ldots, l\}$. In Sovrin, m_1 is reserved for the link secret of the holder, m_2 is reserved for the context – the enumerator for the holders, m_3 is reserved for the policy address I. By default, $\{1, 3\} \subset A_h$ whereas $2 \notin A_h$.

Issuer defines the non-revocation credential with 2 attributes m_1, m_2. In Sovrin, $A_h = \{1\}$ and m_1 is reserved for the link secret of the holder, m_2 is reserved for the context – the enumerator for the holders.

4.2 Primary Credential Cryptographic Setup

In Sovrin, issuers use CL-signatures [**?**] for primary credentials, although other signature types will be supported too.

For the CL-signatures issuer generates:

1. Random 1536-bit primes p', q' such that $p \leftarrow 2p' + 1$ and $q \leftarrow 2q' + 1$ are primes too. Then compute $n \leftarrow pq$.

2. A random quadratic residue S modulo n;

3. Random $x_Z, x_{R_1}, \ldots, x_{R_l} \in [2; p'q' \quad 1]$

Issuer computes

$$Z \leftarrow S^{x_Z} \pmod{n}; \quad \{R_i \leftarrow S^{x_{R_i}} \pmod{n}\}_{1 \le i \le l}; \tag{1}$$

The issuer's public key is $P_k = (n, S, Z, \{R_i\}_{1 \le i \le l})$ and the private key is $s_k = (p, q)$.

4.3 Setup Correctness Proof

4.3.1 Proof: n is a product of two safe primes

1. Let 2^ℓ be an upper-bound on the length of the largest factor of the modulus
2. Let $\epsilon > 1$ be a security parameter
3. A group of prime order $\mathbb{Q} > 2^{2\epsilon\ell+5}$
4. Choose two generators g, h such that $\log_g h$ is not known and computing discrete logarithms is infeasible
5. Compute $c_n := g^n h^{r_n}$ where $r_n \in_R \mathbb{Z}_Q$
6. Compute $c_p := g^p h^{r_p}$, $c_{\tilde{p}} := g^{(p \quad 1)/2} h^{r_{\tilde{p}}}$, $c_q := g^q h^{r_q}$, $c_{\tilde{q}} := g^{(q \quad 1)/2} h^{r_{\tilde{q}}}$ where $r_p, r_{\tilde{p}}, r_q, r_{\tilde{q}}, \in_R \mathbb{Z}_Q$
7. Compute

$$S_{51} := PK\{(\quad, \beta, \gamma, \delta, \rho, \nu, \xi, \chi, \varepsilon, \zeta, \eta) : \tag{2}$$

$$c_{\tilde{p}} = g \;\; h^\beta \;\wedge\; (\;\; 2^{\ddot{\ell}} < \quad < 2^\ell) \wedge \tag{3}$$

$$c_{\tilde{q}} = g^\gamma h^\delta \;\wedge\; (\;\; 2^{\ddot{\ell}} < \quad < 2^\ell) \wedge \tag{4}$$

$$c_p = g^\rho h^\nu \;\wedge\; c_q = g^\xi h^\chi \wedge \tag{5}$$

$$c_p/(c_{\tilde{p}}^2 g) = h^\varepsilon \;\wedge\; c_q/(c_{\tilde{q}}^2 g) = h^\zeta \;\wedge\; c_n/(c_p c_q) = h^\eta \wedge \tag{6}$$

$$\in \texttt{psue oprime(t)} \;\wedge\; \gamma \in \texttt{psue oprime(t)} \wedge \tag{7}$$

$$\rho \in \texttt{psue oprime(t)} \;\wedge\; \xi \in \texttt{psue oprime(t)} \;\} \tag{8}$$

Where `psue oprime(t)` is equal to `t` to the number of bases chosen for Lehmann's primality test.

4.3.2 Lehmann s primality test

Any integer odd integer $n > 1$ is prime if and only if

1. choose k random bases $a_1, ..., a_k \in \mathbb{Z}_n^*$
2. check if $a_i^{(n \quad 1)/2} \equiv \pm 1 \pmod{n}$ holds for all i
3. check of $a_i^{(n \quad 1)/2} \equiv \quad 1 \pmod{n}$ for at least one i

4.3.3 Continuing the correctness proof

1. Issuer generates random $\widetilde{x_Z}, \widetilde{x_{R_1}}, \ldots, \widetilde{x_{R_l}} \in [2; p'q' \quad 1]$;
2. Computes

$$\widetilde{Z} \leftarrow S^{\widetilde{x_Z}} \pmod{n}; \qquad \{\widetilde{R_i} \leftarrow S^{\widetilde{x_{R_i}}} \pmod{n}\}_{1 \le i \le l}; \tag{9}$$

$$c \leftarrow H_I(Z || \widetilde{Z} || \{R_i, \widetilde{R_i}\}_{i \le i \le l}); \tag{10}$$

$$\widehat{x_Z} \leftarrow \widetilde{x_Z} + c x_Z \pmod{p'q'}; \qquad \{\widehat{x_{R_i}} \leftarrow \widetilde{x_{R_i}} + c x_{R_i} \pmod{p'q'}\}_{1 \le i \le l}; \tag{11}$$

   Here H_I is the issuer-defined hash function, by default SHA-256.

3. Proof $\mathcal{P}_I$ of correctness is $(c, \widehat{x_Z}, \{\widehat{x_{R_i}}\}_{1 \le i \le l})$

4.4 Non-revocation Credential Cryptographic Setup

In Sovrin, issuers use CKS accumulator and signatures [**?**] to track revocation status of primary credentials, although other signature types will be supported too. Each primary credential is given an index from 1 to L.

The CKS accumulator is used to track revoked primary credentials, or equivalently, their indices. The accumulator contains up to L indices of credentials. If issuer has to issue more credentials, another accumulator is prepared, and so on. Each accumulator A has an identifier I_A.

Issuer chooses

Groups $_1$, $_2$, $_T$ of prime order q;

Type-3 pairing operation e : $_1 \times \; _2 \rightarrow \; _T$.

Generators: g for $\quad_1$, g' for $\quad_2$.

Issuer:

1. Generates

   1.1. Random $h, h_0, h_1, h_2, \widetilde{h} \in \quad_1$;

   1.2. Random $u, \widehat{h} \in \quad_2$;

   1.3. Random $sk, x \pmod q$.

2. Computes

$$pk \leftarrow g^{sk}; \qquad\qquad y \leftarrow \widehat{h}^{x}.$$

The revocation public key is $P_r = (h, h_0, h_1, h_2, \widetilde{h}, \widehat{h}, u, pk, y)$ and the secret key is (x, sk).

4.4.1 New Accumulator Setup

To create a new accumulator A, issuer:

1. Generates random $\gamma \pmod q$.

2. Computes

   2.1. $g_1, g_2, \ldots, g_L, g_{L+2}, \ldots, g_{2L}$ where $g_i = g^{\gamma^i}$.

   2.2. $g'_1, g'_2, \ldots, g'_L, g'_{L+2}, \ldots, g'_{2L}$ where $g'_i = g'^{\gamma^i}$.

   2.3. $z = (e(g, g'))^{\gamma^{L+1}}$.

3. Set $V \leftarrow \emptyset$, acc $\leftarrow 1$.

The accumulator public key is $P_a = (z)$ and secret key is (γ).
Issuer publishes (P_a, V) on the ledger. The accumulator identifier is $ID_a = z$.

5 Issuance of Credentials

5.1 Holder Setup

Holder:

Loads credential schema $\mathcal{S}$.

Sets hidden attributes $\{m_i\}_{i \in A_h}$.

Establishes a connection with issuer and gets nonce n_0 either from issuer or as a precomputed value. Holder is known to issuer with identifier $\mathcal{H}$.

Holder prepares data for primary credential:

1. Generate random 3152-bit v'.

2. Generate random 593-bit $\{\widetilde{m_i}\}_{i \in A_h}$, and random 3488-bit $\widetilde{v'}$.

3. Compute taking S, Z, R_i from P_k:

$$U \leftarrow (S^{v'}) \prod_{i \in A_h} R_i^{m_i} \pmod n; \tag{12}$$

4. For proving correctness of U, compute

$$\widetilde{U} \leftarrow (S^{\widetilde{v'}}) \prod_{i \in A_h} R_i^{\widetilde{m_i}} \pmod n; \tag{13}$$

$$c \leftarrow H(U||\widetilde{U}||n_0); \qquad\qquad \widehat{v'} \leftarrow \widetilde{v'} + cv'; \tag{14}$$

$$\{\widehat{m_i} \leftarrow \widetilde{m_i} + cm_i\}_{i \in A_h}; \tag{15}$$

5. Generate random 80-bit nonce n_1

6. Send $\{U, c, \widehat{v'}, \{\widehat{m_i}\}_{i \in A_h}, n_1\}$ to the issuer.

Holder prepares for non-revocation credential:

1. Load issuer's revocation key P_R and generate random $s'_R \bmod q$.
2. Compute $U_R \leftarrow h_2^{s'_R}$ taking h_2 from P_R.
3. Send U_R to the issuer.
4. For proving correctness of U_R

   generate random $\widetilde{s'_R} \bmod q$ and compute $\widetilde{U_R} \leftarrow h_2{}^{\widetilde{s'_R}}$

   Compute above challenge c as $c \leftarrow H(U||\widetilde{U}||U_R||\widetilde{U_R}||n_0)$ instead of $c \leftarrow H(U||\widetilde{U}||n_0)$

   Compute $\widehat{s'_R} \leftarrow \widetilde{s'_R} + cs'_R$

   Send c and $\widehat{s'_R}$ to issuer

5.1.1 Issuer Proof of Setup Correctness

To verify the proof $\mathcal{P}_i$ of correctness, holder computes

$$S_{51} := PK\{(\ ,\beta,\gamma,\delta,\rho,\nu,\xi,\chi,\varepsilon,\zeta,\eta) : \tag{16}$$

$$c_{\tilde{p}} = g\ \ h^{\beta}\ \wedge\ (\ \ 2^{\ddot{\ell}} < \ \ < 2^{\ell}) \wedge \tag{17}$$

$$c_{\tilde{q}} = g^{\gamma}h^{\delta}\ \wedge\ (\ \ 2^{\ddot{\ell}} < \ \ < 2^{\ell}) \wedge \tag{18}$$

$$c_p = g^{\rho}h^{\nu}\ \wedge\ c_q = g^{\xi}h^{\chi} \wedge \tag{19}$$

$$c_p/(c_{\tilde{p}}^2 g) = h^{\varepsilon}\ \wedge\ c_q/(c_{\tilde{q}}^2 g) = h^{\zeta}\ \wedge\ c_n/(c_p c_q) = h^{\eta} \wedge \tag{20}$$

$$\in \texttt{psue oprime(t)}\ \wedge\ \gamma \in \texttt{psue oprime(t)} \wedge \tag{21}$$

$$\rho \in \texttt{psue oprime(t)}\ \wedge\ \xi \in \texttt{psue oprime(t)}\ \} \tag{22}$$

where t is the bases used in the the Lehmann-primality tests.

and then computes

$$\widehat{Z} \leftarrow Z\ \ {}^{c}S^{\widehat{xz}} \pmod n; \quad \{\widehat{R_i} \leftarrow R_i\ {}^{c}S^{\widehat{xR_i}} \pmod n\}_{1 \le i \le l};$$

and verifies

$$c = H_I(Z||\widehat{Z}||\{R_i, \widehat{R_i}\}_{1 \le i \le l})$$

.

5.2 Primary Credential Issuance

Issuer verifies the correctness of holder's input:

1. Compute

$$\widehat{U} \leftarrow (U\ \ {}^{c}) \prod_{i \in A_h} R_i^{\widehat{m_i}}(S^{v'}) \pmod n; \tag{23}$$

2. Verify $c = H(U||\widehat{U}||n_0)$
3. Verify that $\widehat{v'}$ is a 673-bit number, $\{\widehat{m_i}, \widehat{r_i}\}_{i \in \mathcal{A}_c}$ are 594-bit numbers.
4. If a revocable credential is requested

   Compute $\widehat{U_R} = U_R\ \ {}^{c}h_2{}^{\widehat{s'_R}}$

   Verify that c equals $H(U||\widehat{U}||U_R||\widehat{U_R}||n_0)$ instead of $H(U||\widehat{U}||n_0)$

Issuer prepare the credential:

1. Assigns index $i < L$ to holder, which is one of not yet taken indices for the issuer's current accumulator A. Compute $m_2 \leftarrow H(i||\mathcal{H})$ and store information about holder and the value i in a local database.

2. Set, possibly in agreement with holder, the values of disclosed attributes, i.e. with indices from A_k.

3. Generate random 2724-bit number v'' with most significant bit equal 1 and random prime e such that

$$2^{596} \leq e \leq 2^{596} + 2^{119}. \tag{24}$$

4. Compute

$$Q \leftarrow \frac{Z}{US^{v''} \prod_{i \in \mathcal{A}_k} R_i^{m_i} \pmod n}; \tag{25}$$

$$A \leftarrow Q^{e^{-1} \pmod{p'q'}} \pmod n; \tag{26}$$

5. Generate random $r < p'q'$;

6. Compute

$$\widehat{A} \leftarrow Q^r \pmod n; \tag{27}$$

$$c' \leftarrow H(Q||A||\widehat{A}||n_1); \tag{28}$$

$$s_e \leftarrow r - c'e^{-1} \pmod{p'q'}; \tag{29}$$

7. Send the primary pre-credential $(\{m_i\}_{i \in A_k}, A, e, v'', s_e, c')$ to the holder.

5.3 Non-revocation Credential Issuance

Issuer:

1. Generate random numbers $s'', c \bmod q$.

2. Take m_2 from the primary credential he is preparing for holder.

3. Take A as the accumulator value for which index i was taken. Retrieve current set of non-revoked indices V.

4. Compute:

$$\sigma \leftarrow \left(h_0 h_1^{m_2} \cdot U_R \cdot g_i \cdot h_2^{s''}\right)^{\frac{1}{x+c}}; \qquad w \leftarrow \prod_{j \in V} g'_{L+1-j+i}; \tag{30}$$

$$\sigma_i \leftarrow g'^{1/(sk+\gamma^i)}; \qquad u_i \leftarrow u^{\gamma^i}; \tag{31}$$

$$A \leftarrow A \cdot g'_{L+1-i}; \qquad V \leftarrow V \cup \{i\}; \tag{32}$$

$$\text{wit}_i \leftarrow \{\sigma_i, u_i, g_i, w, V\}. \tag{33}$$

5. Send the non-revocation pre-credential $(I_A, \sigma, c, s'', \text{wit}_i, g_i, g'_i, i)$ to holder.

6. Publish updated V, A on the ledger.

5.4 Storing Credentials

Holder works with the primary pre-credential :

1. Compute $v \leftarrow v' + v''$.

2. Verify e is prime and satisfies Eq. (24).

3. Compute

$$Q \leftarrow \frac{Z}{S^v \prod_{i \in C_s} R_i^{m_i}} \pmod n; \tag{34}$$

4. Verify $Q = A^e \pmod n$

5. Compute [2]

$$\widehat{A} \leftarrow A^{c' + s_e \cdot e} \pmod n. \tag{35}$$

[2]We have removed factor $S^{v' s_e}$ here from computing of as it seems to be a typo in the Idemix spec.

6. Verify $c' = H(Q||A||\widehat{A}||n_2)$.

7. Store *primary credential* $C_p = (\{m_i\}_{i \in C_s}, A, e, v)$.

Holder takes the non-revocation pre-credential $(I_A, \sigma, c, s'', \text{wit}_i, g_i, g'_i, i)$ computes $s_R \leftarrow s' + s''$ and stores the non-revocation credential $C_{NR} \leftarrow (I_A, \sigma, c, s, \text{wit}_i, g_i, g'_i, i)$.

5.5 Non revocation proof of correctness

Holder computes

$$\frac{e(g_i, acc_V)}{e(g, w)} \stackrel{?}{=} z; \tag{36}$$

$$e(pk \cdot g_i, \sigma_i) \stackrel{?}{=} e(g, g'); \tag{37}$$

$$e(\sigma, y \cdot \widehat{h}^c) \stackrel{?}{=} e(h_0 \cdot h_1^{m_2} h_2^s g_i, \widehat{h}). \tag{38}$$

6 Revocation

Issuer identifies a credential to be revoked in the database and retrieves its index i, the accumulator value A, and valid index set V. Then he proceeds:

1. Set $V \leftarrow V \setminus \{i\}$;

2. Compute $A \leftarrow A / g'_{L+1 \quad i}$.

3. Publish $\{V, A\}$.

7 Presentation

7.1 Proof Request

Verifier sends a proof request, where it specifies the ordered set of d credential schemas $\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_d\}$, so that the holder should provide a set of d credential pairs (C_p, C_{NR}) that correspond to these schemas.

Let credentials in these schemas contain X attributes in total. Suppose that the request makes to open x_1 attributes, makes to prove x_2 equalities $m_i = m_j$ (from possibly distinct schemas) and makes to prove x_3 predicates of form $m_i >\leq\geq< z$. Then effectively $X \quad x_1$ attributes are unknown (denote them A_h), which form $x_4 = (X \quad x_1 \quad x_2)$ equivalence classes. Let ϕ map A_h to $\{1, 2, \ldots, x_4\}$ according to this equivalence. Let A_v denote the set of indices of x_1 attributes that are disclosed.

The proof request also specifies A_h, ϕ, A_v and the set $\mathcal{D}$ of predicates. Along with a proof request, Verifier also generates and sends 80-bit nonce n_1.

7.2 Proof Preparation

Holder prepares all credential pairs (C_p, C_{NR}) to submit:

1. Generates x_4 random 592-bit values $\widetilde{y_1}, \widetilde{y_2}, \ldots, \widetilde{y_{x_4}}$ and set $\widetilde{m_j} \leftarrow \widetilde{y_{\phi(j)}}$ for $j \in \mathcal{A}_h$.

2. Create empty sets $\mathcal{T}$ and $\mathcal{C}$.

3. For all credential pairs (C_p, C_{NR}) executes Section 7.2.

4. Executes Section 7.2.1 once.

5. For all credential pairs (C_p, C_{NR}) executes Section 7.2.2.

6. Executes Section 7.2.2 once.

Verifier:

1. For all credential pairs (C_p, C_{NR}) executes Section 7.3.

2. Executes Section 7.3.3 once.

Non-revocation proof Holder:

1. Load issuer's public revocation key $p = (h, h_1, h_2, \widetilde{h}, \widehat{h}, u, pk, y)$.
2. Load the non-revocation credential $C_{NR} \leftarrow (I_A, \sigma, c, s, \text{wit}_i, g_i, g'_i, i)$;
3. Obtain recent V, acc (from Verifier, Sovrin link, or elsewhere).
4. Update C_{NR}:

$$w \leftarrow w \cdot \frac{\prod_{j \in V \setminus V_{old}} g'_{L+1 \quad j+i}}{\prod_{j \in V_{old} \setminus V} g'_{L+1 \quad j+i}};$$
$$V_{old} \leftarrow V.$$

   Here V_{old} is taken from wit_i and updated there.
5. Select random $\rho, \rho', r, r', r'', r''', o, o' \bmod q$;
6. Compute

$$E \leftarrow h^{\rho} \widetilde{h}^{o} \qquad D \leftarrow g^{r} \widetilde{h}^{o'}; \tag{39}$$
$$A \leftarrow \sigma \widetilde{h}^{\rho} \qquad \mathcal{G} \leftarrow g_i \widetilde{h}^{r}; \tag{40}$$
$$\mathcal{W} \leftarrow w \widehat{h}^{r'} \qquad \mathcal{S} \leftarrow \sigma_i \widehat{h}^{r''} \tag{41}$$
$$\mathcal{U} \leftarrow u_i \widehat{h}^{r'''} \tag{42}$$

   and adds these values to $\mathcal{C}$.
7. Compute

$$m \leftarrow \rho \cdot c \mod q; \qquad t \leftarrow o \cdot c \mod q; \tag{43}$$
$$m' \leftarrow r \cdot r'' \mod q; \qquad t' \leftarrow o' \cdot r'' \mod q; \tag{44}$$

   and adds these values to $\mathcal{C}$.
8. Generate random $\widetilde{\rho}, \widetilde{o}, \widetilde{o'}, \widetilde{c}, \widetilde{m}, \widetilde{m'}, \widetilde{t}, \widetilde{t'}, \widetilde{m_2}, \widetilde{s}, \widetilde{r}, \widetilde{r'}, \widetilde{r''}, \widetilde{r'''}, \bmod q$.
9. Compute

$$\overline{T_1} \leftarrow h^{\widetilde{\rho}} \widetilde{h}^{\widetilde{o}} \qquad \overline{T_2} \leftarrow E^{\widetilde{c}} h^{\ \widetilde{m}} \widetilde{h}^{\ \widetilde{t}} \tag{45}$$

$$\overline{T_3} \leftarrow e(A, \widehat{h})^{\widetilde{c}} \cdot e(\widetilde{h}, \widehat{h})^{\widetilde{r}} \cdot e(\widetilde{h}, y)^{\ \widetilde{\rho}} \cdot e(\widetilde{h}, \widehat{h})^{\ \widetilde{m}} \cdot e(h_1, \widehat{h})^{\ \widetilde{m_2}} \cdot e(h_2, \widehat{h})^{\ \widetilde{s}} \tag{46}$$

$$\overline{T_4} \leftarrow e(\widetilde{h}, \text{acc})^{\widetilde{r}} \cdot e(1/g, \widehat{h})^{\widetilde{r'}} \qquad \overline{T_5} \leftarrow g^{\widetilde{r}} \widetilde{h}^{\widetilde{o'}} \tag{47}$$
$$\overline{T_6} \leftarrow D^{\widetilde{r''}} g^{\ \widetilde{m'}} \widetilde{h}^{\ \widetilde{t'}} \qquad \overline{T_7} \leftarrow e(pk \cdot \mathcal{G}, \widehat{h})^{\widetilde{r''}} \cdot e(\widetilde{h}, \widehat{h})^{\ \widetilde{m'}} \cdot e(\widetilde{h}, \mathcal{S})^{\widetilde{r}} \tag{48}$$
$$\overline{T_8} \leftarrow e(\widetilde{h}, u)^{\widetilde{r}} \cdot e(1/g, \widehat{h})^{\widetilde{r'''}} \tag{49}$$

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

Validity proof

Holder:

1. Generate a random 592-bit number $\widetilde{m_j}$ for each $j \in \mathcal{A}_{\overline{r}}$.
2. For each credential $C_p = (\{m_j\}, A, e, v)$ and issuer's public key pk_I:
   2.1. Choose random 3152-bit r.
   2.2. Take n, S from pk_I compute

$$A' \leftarrow AS^{r} \pmod n \text{ and } v' \leftarrow v \quad e \cdot r \text{ as integers;} \tag{50}$$

   and add to $\mathcal{C}$.
   2.3. Compute $e' \leftarrow e \quad 2^{596}$.
   2.4. Generate random 456-bit number $\widetilde{e}$.

2.5. Generate random 3748-bit number $\widetilde{v}$.

2.6. Compute

$$T \leftarrow (A')^{\widetilde{e}} \left(\prod_{j \in \mathcal{A}_{\overline{r}}} R_j^{\widetilde{m_j}} \right) (S^{\widetilde{v}}) \pmod{n} \tag{51}$$

and add to $\mathcal{T}$.

3. Load Z, S from issuer's public key.

4. For each predicate p where the operator $*$ is one of $>, \geq, <, \leq$.

4.1. Calculate such that:

$$\leftarrow \begin{cases} z_j \quad m_j; & \text{if } * \equiv \leq \\ z_j \quad m_j \quad 1; & \text{if } * \equiv < \\ m_j \quad z_j; & \text{if } * \equiv \geq \\ m_j \quad z_j \quad 1; & \text{if } * \equiv > \end{cases}$$

4.2. Calculate a such that:

$$a \leftarrow \begin{cases} 1 & \text{if } * \equiv \leq or < \\ 1 & \text{if } * \equiv \geq or > \end{cases}$$

4.3. Find (possibly by exhaustive search) u_1, u_2, u_3, u_4 such that:

$$= (u_1)^2 + (u_2)^2 + (u_3)^2 + (u_4)^2 \tag{52}$$

4.4. Generate random 2128-bit numbers r_1, r_2, r_3, r_4, r .

4.5. Compute

$$\{T_i \leftarrow Z^{u_i} S^{r_i} \pmod{n}\}_{1 \leq i \leq 4}; \tag{53}$$

$$T \leftarrow Z \quad S^r \pmod{n}; \tag{54}$$

and add these values to $\mathcal{C}$ in the order T_1, T_2, T_3, T_4, T .

4.6. Generate random 592-bit numbers $\widetilde{u_1}, \widetilde{u_2}, \widetilde{u_3}, \widetilde{u_4}$.

4.7. Generate random 672-bit numbers $\widetilde{r_1}, \widetilde{r_2}, \widetilde{r_3}, \widetilde{r_4}, \widetilde{r^{\sim}}$.

4.8. Generate random 2787-bit number $\widetilde{\ }$

4.9. Compute

$$\{\overline{T_i} \leftarrow Z^{\widetilde{u_i}} S^{\widetilde{r_i}} \pmod{n}\}_{1 \leq i \leq 4}; \tag{55}$$

$$\overline{T^{\sim}} \leftarrow Z^{\widetilde{m_j}} S^{\widetilde{ar^{\sim}}} \pmod{n}; \tag{56}$$

$$Q \leftarrow (S^{\widetilde{\ }}) \prod_{i=1}^{4} T_i^{\widetilde{u_i}} \pmod{n}; \tag{57}$$

and add these values to $\mathcal{T}$ in the order $\overline{T_1}, \overline{T_2}, \overline{T_3}, \overline{T_4}, \overline{T^{\sim}}, Q$.

7.2.1 Hashing

Holder computes challenge hash

$$c_H \leftarrow H(\mathcal{T}, \mathcal{C}, n_1); \tag{58}$$

and sends c_H to Verifier.

7.2.2 Final preparation

Holder:

1. For non-revocation credential C_{NR} compute:

$$\widehat{\rho} \leftarrow \widetilde{\rho} \quad c_H\rho \bmod q \qquad \widehat{o} \leftarrow \widetilde{o} \quad c_H \cdot o \bmod q$$
$$\widehat{c} \leftarrow \widetilde{c} \quad c_H \cdot c \bmod q \qquad \widehat{o'} \leftarrow \widetilde{o'} \quad c_H \cdot o' \bmod q$$
$$\widehat{m} \leftarrow \widetilde{m} \quad c_H m \bmod q \qquad \widehat{m'} \leftarrow \widetilde{m'} \quad c_H m' \bmod q$$
$$\widehat{t} \leftarrow \widetilde{t} \quad c_H t \bmod q \qquad \widehat{t'} \leftarrow \widetilde{t'} \quad c_H t' \bmod q$$
$$\widehat{m_2} \leftarrow \widetilde{m_2} \quad c_H m_2 \bmod q \qquad \widehat{s} \leftarrow \widetilde{s} \quad c_H s \bmod q$$
$$\widehat{r} \leftarrow \widetilde{r} \quad c_H r \bmod q \qquad \widehat{r'} \leftarrow \widetilde{r'} \quad c_H r' \bmod q$$
$$\widehat{r''} \leftarrow \widetilde{r''} \quad c_H r'' \bmod q \qquad \widehat{r'''} \leftarrow \widetilde{r'''} \quad c_H r''' \bmod q.$$

   and add them to $\mathcal{X}$.

2. For primary credential C_p compute:

$$\widehat{e} \leftarrow \widetilde{e} + c_H e'; \tag{59}$$
$$\widehat{v} \leftarrow \widetilde{v} + c_H v'; \tag{60}$$
$$\{\widehat{m}_j \leftarrow \widetilde{m_j} + c_H m_j\}_{j \in \mathcal{A}_{\overline{r}}}; \tag{61}$$

   The values $Pr_C = (\widehat{e}, \widehat{v}, \{\widehat{m_j}\}_{j \in \mathcal{A}_{\overline{r}}}, A')$ are the *sub-proof* for credential C_p.

3. For each predicate p compute:

$$\{\widehat{u_i} \leftarrow \widetilde{u_i} + c_H u_i\}_{1 \le i \le 4}; \tag{62}$$
$$\{\widehat{r_i} \leftarrow \widetilde{r_i} + c_H r_i\}_{1 \le i \le 4}; \tag{63}$$
$$\widehat{r^{\frown}} \leftarrow \widetilde{r^{\frown}} + c_H r \ ; \tag{64}$$
$$\widehat{\ ^{\frown}} \leftarrow \widetilde{\ ^{\frown}} + c_H(r \quad u_1 r_1 \quad u_2 r_2 \quad u_3 r_3 \quad u_4 r_4); \tag{65}$$

   The values $Pr_p = (\{\widehat{u_i}\}, \{\widehat{r_i}\}, \widehat{r^{\frown}}, \widehat{\ ^{\frown}}, \widehat{m_j})$ are the sub-proof for predicate p.

7.2.3 Sending

Holder sends $(c_H, \mathcal{X}, \{Pr_C\}, \{Pr_p\}, \mathcal{C})$ to the Verifier.

7.3 Veri cation

For the credential pair (C_p, C_{NR}), Verifier retrieves relevant variables from $\mathcal{X}, \{Pr_C\}, \{Pr_p\}, \mathcal{C}$.

7.3.1 Non-revocation check

Verifier computes

$$\widehat{T_1} \leftarrow E^c \ \cdot h^{\rho} \cdot \widetilde{h}^{o} \qquad \widehat{T_2} \leftarrow E^c \cdot h^{\ m} \cdot \widetilde{h}^{\ t} \tag{66}$$

$$\widehat{T_3} \leftarrow \left(\frac{e(h_0\mathcal{G}, \widehat{h})}{e(A, y)}\right)^c \cdot e(A, \widehat{h})^c \cdot e(\widetilde{h}, \widehat{h})^r \cdot e(\widetilde{h}, y)^{\ \rho} \cdot e(\widetilde{h}, \widehat{h})^{\ m} \cdot e(h_1, \widehat{h})^{\ \widehat{m_2}} \cdot e(h_2, \widehat{h})^{\ s} \tag{67}$$

$$\widehat{T_4} \leftarrow \left(\frac{e(\mathcal{G}, \mathrm{acc})}{e(g, \mathcal{W})z}\right)^c \cdot e(\widetilde{h}, \mathrm{acc})^r \cdot e(1/g, \widehat{h})^{r'} \qquad \widehat{T_5} \leftarrow D^c \ \cdot g^r \widetilde{h}^{o'} \tag{68}$$

$$\widehat{T_6} \leftarrow D^{\widehat{r''}} \cdot g^{\ \widehat{m'}} \widetilde{h}^{\ t'} \qquad \widehat{T_7} \leftarrow \left(\frac{e(pk \cdot \mathcal{G}, \mathcal{S})}{e(g, g')}\right)^c \cdot e(pk \cdot \mathcal{G}, \widehat{h})^{\widehat{r''}} \cdot e(\widetilde{h}, \widehat{h})^{\ \widehat{m'}} \cdot e(\widetilde{h}, \mathcal{S})^r \tag{69}$$

$$\widehat{T_8} \leftarrow \left(\frac{e(\mathcal{G}, u)}{e(g, \mathcal{U})}\right)^c \cdot e(\widetilde{h}, u)^r \cdot e(1/g, \widehat{h})^{\widehat{r'''}} \tag{70}$$

and adds these values to $\widehat{T}$.

7.3.2 Validity

Verifier uses all issuer public key pk_I involved into the credential generation and the received $(c, \widehat{e}, \widehat{v}, \{\widehat{m_j}\}, A')$. He also uses revealed $\{m_j\}_{j \in \mathcal{A}_r}$. He initiates $\widehat{\mathcal{T}}$ as empty set.

1. For each credential C_p, take each sub-proof Pr_C and compute

$$\widehat{T} \leftarrow \left(\frac{Z}{\prod_{j \in \mathcal{A}_r} {R_j}^{m_j}) \, (A')^{2^{596}}} \right)^{c} (A')^{e} \left(\prod_{j \in (\mathcal{A}_{\bar{r}})} {R_j}^{\widehat{m_j}} \right) (S^{v}) \pmod{n}. \tag{71}$$

   Add $\widehat{T}$ to $\widehat{\mathcal{T}}$.

2. For each predicate p:

$$' \leftarrow \begin{cases} z_j; & \text{if } * \equiv \leq \\ z_j \quad 1; & \text{if } * \equiv < \\ z_j; & \text{if } * \equiv \geq \\ z_j + 1; & \text{if } * \equiv > \end{cases}$$

$$a \leftarrow \begin{cases} 1 & \text{if } * \equiv \leq \textit{ or } < \\ 1 & \text{if } * \equiv \geq \textit{ or } > \end{cases}$$

   2.1. Using Pr_p and $\mathcal{C}$ compute

$$\{\widehat{T_i} \leftarrow T_i \; {}^{c} Z^{u_i} S^{r_i} \pmod{n}\}_{1 \leq i \leq 4}; \tag{72}$$

$$\widehat{T^{\,}} \leftarrow \; T^{a} Z^{\;\prime} \Big)^{\;c} Z^{\widehat{m_j}} S^{\widehat{a r^{\,}}} \pmod{n}; \tag{73}$$

$$\widehat{Q} \leftarrow (T^{\;\;c}) \prod_{i=1}^{4} T_i^{u_i} (S^{\;\;}) \pmod{n}, \tag{74}$$

   and add these values to $\widehat{\mathcal{T}}$ in the order $\widehat{T_1}, \widehat{T_2}, \widehat{T_3}, \widehat{T_4}, \widehat{T^{\,}}, \widehat{Q}$.

7.3.3 Final hashing

1. Verifier computes

$$\widehat{c_H} \leftarrow H(\widehat{\mathcal{T}}, \mathcal{C}, n_1).$$

2. If $c = \widehat{c}$ output VERIFIED else FAIL.

8 Changelog

8.1 25 April 2022 (version 0.6)

Proof that n is a product of two safe primes in Section 4.3.1

Issuer correctness proof no longer considered optional

8.2 20 Jun 2019 (version 0.5)

Holder's proof of correctness for revocation secret

Fixing several typos

8.3 9 Feb 2018 (version 0.4)

Formatting and updates for committed attributes

8.4 7 Feb 2018 (version 0.3)

Type-3-pairing-based revocation added.

8.5 7 Feb 2018 (version 0.21)

c changed to c in Section 5, item 1.0.1.

Factor $S^{v's_e}$ is removed from item 3.2.0.

8.6 13 July 2017

Added:

Proof of correctness for issuer's setup in Section 4;

Verification of correctness of setup: steps 1.0.1, 1.0.2;

Proof of correctness for holder's blinded attributes: steps 1.3.1, 1.3.2, 1.4;

Verification holder's proof of correctness: steps 2.0.1, 2.0.2;

Issuer sends all m_i in step 2.4.

Proof of correctness for issuer's signature: steps 2.2.1, 2.2.2, 2.2.3.

Verification of correctness of signature: steps 3.1.0, 3.1.1, 3.1.2, 3.2.0, 3.2.1.