

Anonymous credentials with type-3 revocation

Dmitry Khovratovich

7 February 2018, version 0.3

1 Introduction

1.1 Concept

The concept of *anonymous credentials* allows users of certain web service (for example, online banking) to prove that their identity satisfy certain properties in uncorrelated way without revealing the other identity details. The properties can be raw identity attributes such as the birth date or the address, or a more sophisticated predicates such as “A is older than 20 years old”.

We assume three parties: Issuer, Prover, Verifier (Service). From the functional perspective, the Issuer gives a credential C based on identity X , which asserts certain property \mathcal{P} about X , to the Prover. The identity consists of attributes m_1, m_2, \dots, m_l . The Prover then presents (\mathcal{P}, C) to the Verifier, which can verify that the Issuer had checked that Prover’s identity has property \mathcal{P} .

For compliance, another party Inspector is often deployed. Inspector is able to deanonymize the Prover given the transcript of his interaction with the Verifier.

1.2 Properties

First of all, 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. Technically it is implemented by the Prover *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-show* so that second and later presentations are detected.

Credentials are *delegatable* if Prover A can delegate a credential C to Prover B with certain attributes X , so that Verifier would not learn the identity of A if B presents Y to him. The delegation may continue further thus creating a credential chain.

1.3 Pseudonyms

Typically a credential is bound to a certain pseudonym nym . It is supposed that Prover 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 Prover 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 master secret m_1 together with nym in a way that nym can not be linked to nym_V . However, Prover is supposed to prove that the credential he presents was issued to a pseudonym derived from the same master secret as used to produce nym_V .

2 Simple example

Government (Issuer G) issues credentials with age and photo hash. Company ABC-Co (Issuer A) issues a credential that includes start-date and employment status, e.g., ‘FULL-TIME’.

Prover establishes a pseudonym with both issuers. Then he got two credentials independently. After that he proves to Verifier that he has two credentials such that

- The same master secret is used in both credentials;
- The age value in the first credential is over 20;

- The employment status is 'FULL-TIME'.

Steps in Section 4.2 must be executed for each Issuer.

Issuer and Prover mutually trust each other in submitting values of the right format during credential's issuance. This trust can be eliminated at the cost of some extra steps.

3 Functionality

In this description of the protocol Prover has only one attribute he hides from Issuer – his master secret m_1 .

4 Setup

This section describes how the Prover obtains the primary claim and the non-revocation claim from the Issuer.

If there are multiple Issuers, the Prover obtains the claims from them independently. If the Prover wants to present the claims together and link them, he should *chain* the claims by ensuring that every two consecutive primary claims in the chain share at least one attribute of the same value. The simplest way to ensure that is to generate a single master key K and use it as the value of the first attribute m_1 in all primary claims.

4.1 Prover setup

Prover generates 256-bit master key K (possibly the same for all Issuers) and set $m_1 = K$.

4.2 Issuer setup

Issuer defines the primary claim schema P with l attributes m_1, m_2, \dots, m_l . The m_1 is reserved for the master secret of the Prover and m_2 is reserved for the context – the enumerator for the provers.

Primary claim setup:

1. Generate random 1024-bit primes p', q' such that $p \leftarrow 2p' + 1$ and $q \leftarrow 2q' + 1$ are primes too. Finally compute $n \leftarrow pq$.
2. Generate a random quadratic residue¹ S modulo n ;
3. Select random $x_Z, x_{R_1}, \dots, x_{R_l} \in [2; p'q' - 1]$ and compute $Z \leftarrow S^{x_Z} \pmod{n}$, $R_i \leftarrow S^{x_{R_i}} \pmod{n}$ for $1 \leq i \leq l$.

The issuer's public key is $pk_I = (n, S, Z, R_1, R_2, \dots, R_l, P)$ and the private key is $sk_I = (p, q)$.

Setup correctness:

1. Generate random $\widetilde{x}_Z, \widetilde{x}_{R_1}, \dots, \widetilde{x}_{R_l} \in [2; p'q' - 1]$;
2. Compute

$$\widetilde{Z} \leftarrow S^{\widetilde{x}_Z} \pmod{n}, \quad \widetilde{R}_i \leftarrow S^{\widetilde{x}_{R_i}} \pmod{n}, 1 \leq i \leq l; \quad (1)$$

$$c \leftarrow H(Z || R_1 || \dots || R_l || \widetilde{Z} || \widetilde{R}_1 || \dots || \widetilde{R}_l); \quad (2)$$

$$\widehat{x}_Z \leftarrow \widetilde{x}_Z + c \cdot x_Z, \quad \widehat{x}_{R_i} \leftarrow \widetilde{x}_{R_i} + c \cdot x_{R_i}, 1 \leq i \leq l. \quad (3)$$

3. Proof P_I of correctness is $(c, \widehat{x}_Z, \{\widehat{x}_{R_i}\}_i)$

Non-revocation claim setup: Common parameters:

- Groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of prime order q ;
- Type-3 pairing operation $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$.
- Generators: g for \mathbb{G}_1 , g' for \mathbb{G}_2 .

Typically the triplet $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ is selected together with a pairing function as only a few combinations admit a suitable pairing. Existing implementations provide just a few possible pairing functions and thus triplets, thus making the group details in fact oblivious to the user. For the sake of curiosity we note that $\mathbb{G}_1, \mathbb{G}_2$ are different groups of elliptic curve points, whereas \mathbb{G}_T is not a curve point group.

Issuer makes the following steps:

¹using the function *randomQR* from the Charm library.

1. Generate random $h, h_0, h_1, h_2, \tilde{h} \in \mathbb{G}_1$;
2. Generate random $u, \hat{h} \in \mathbb{G}_2$;
3. Generate random $sk, x \pmod{q}$.
4. Compute

$$pk \leftarrow g^{sk}; \quad y \leftarrow \hat{h}^x.$$

5. The issuer revocation public key is $pk^R = (h, h_0, h_1, h_2, \tilde{h}, \hat{h}, u, pk, y)$ and the secret key is (x, sk) .

The Issuer fixes the number L of credentials per accumulator. For each accumulator:

1. Generate random $\gamma \pmod{q}$.
2. Compute $g_1, g_2, \dots, g_L, g_{L+2}, \dots, g_{2L}$ where $g_i = g^{\gamma^i}$.
3. Compute $g'_1, g'_2, \dots, g'_L, g'_{L+2}, \dots, g'_{2L}$ where $g'_i = g^{\gamma^i}$.
4. Compute $z = (e(g, g'))^{\gamma^{L+1}}$.
5. Set $V \leftarrow \emptyset$, $\text{acc} \leftarrow 1$.

The accumulator public key is (z) and secret key is (γ) .

5 Issuance of claims

For the new user Issuer selects the accumulator index A_i and the user index i so that (A_i, i) is unique.

5.1 Primary claim

- 0.1 Issuer retrieves the current value acc for accumulator A_i and the set V of issued and non-revoked credential numbers.

- 0.2 Issuer computes

$$\bar{S} = A_i || U_i, \quad H_{\bar{S}} = H(\bar{S}) \pmod{2^{256}}.$$

where U_i is the identifier of the user i . and sets $m_2 = H_{\bar{S}}$.

- 0.3 Issuer sets 256-bit integer attributes m_3, \dots, m_l for the Prover.

- 0.4 Issuer generates 80-bit nonce n_0 and sends it to the Prover;

- 1.0.1 Prover retrieves Issuer's proof of correctness and computes

$$\widehat{Z} \leftarrow Z^{-c} S^{\widehat{xz}} \pmod{n}, \quad \widehat{R}_i \leftarrow R_i^{-c} S^{\widehat{xR_i}} \pmod{n}, 1 \leq i \leq l;$$

- 1.0.2 Prover verifies that

$$c = H(Z || R_1 || \dots || R_l || \widehat{Z} || \widehat{R}_1 || \dots || \widehat{R}_l)$$

- 1.1 Prover generates random 2128-bit v' and loads Issuer's key pk_I .

- 1.2 Prover computes $U \leftarrow S^{v'} R_1^{m_1} \pmod{n}$ taking S from pk_I .

- 1.3.1 Prover generates random 593-bit \widetilde{m}_1 and random 673-bit \widetilde{v}' .

- 1.3.2 Prover computes

$$\widetilde{U} \leftarrow R_1^{\widetilde{m}_1} S^{\widetilde{v}'}; \quad c \leftarrow H(U || \widetilde{U} || n_0); \quad (4)$$

$$\widehat{v}' \leftarrow \widetilde{v}' + c \cdot v'; \quad \widehat{m}_1 \leftarrow \widetilde{m}_1 + c \cdot m_1. \quad (5)$$

- 1.4 Prover generates random 80-bit n_2 and sends $U, P = \{c, \widehat{v}', \widehat{m}_1\}, n_2$ to the Issuer.

2.0.1 Issuer computes

$$\widehat{U} \leftarrow U^{-c} S^{\widehat{v}'} R_1^{\widehat{m}_1}$$

2.0.2 Issuer verifies that

$$c = H(U || \widehat{U} || n_0)$$

2.0.3 Issuer verifies that \widehat{v}' is 673-bit number, \widehat{m}_1 is 594-bit number.

2.1 Issuer generates random 2724-bit number v'' with most significant bit equal 1 and random prime e such that

$$2^{596} \leq e \leq 2^{596} + 2^{119}. \quad (6)$$

2.2.1 Issuer computes

$$Q \leftarrow \frac{Z}{U S^{v''} (R_2^{m_2} R_3^{m_3} \dots R_l^{m_l})} \pmod{n}.$$

and

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

2.2.2 Issuer generates random r smaller than $p'q'$;

2.2.3 Issuer computes

$$\widehat{A} \leftarrow Q^r \pmod{n}; \quad c \leftarrow H(Q || A || \widehat{A} || n_2); \quad s_e \leftarrow r - c' e^{-1} \pmod{p'q'}.$$

2.3 Issuer sends $(\{m_i\}_{i \geq 2}, A, e, v'', P_2 = \{s_e, c'\})$ to the Prover.

3.0 Prover computes $v \leftarrow v' + v''$.

3.1.0 Prover checks that e is prime and satisfies Eq. (6).

3.1.1 Prover computes

$$Q \leftarrow \frac{Z}{S^v \prod_{i \leq l} R_i^{m_i}} \pmod{n}.$$

3.1.2 Prover tests that $Q = A^e \pmod{n}$

3.2.0 Prover retrieves P_2 and computes ²

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

3.2.1 Prover verifies that

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

Prover stores *primary claim* $C_1 = (\{m_i\}, A, e, v)$.

5.2 Non-revocation claim

Prover starts:

1. Loads Issuer's revocation key pk^R and generates random $s' \pmod{q}$.
2. Computes $U \leftarrow h_2^{s'}$ taking h_2 from pk^R .
3. Sends U to the Issuer.

We assume that the attribute m_2 is used to enumerate provers by Issuer (details are irrelevant for revocation). Then Issuer proceeds:

1. Generates random numbers $s'', c \pmod{q}$.
2. Takes m_2 from the primary claim he is preparing for the Prover.
3. Selects the accumulator index A_i and the user index i for the Prover so that i has not been assigned yet for A_i .

²We have removed factor $S^{v's_e}$ here from computing of \widehat{A} as it seems to be a typo in the original paper.

4. Computes

$$\sigma \leftarrow \left(h_0 h_1^{m_2} \cdot U \cdot g_i \cdot h_2^{s''} \right)^{\frac{1}{x+c}}; \quad w \leftarrow \prod_{j \in V} g'_{L+1-j+i}; \quad (7)$$

$$\sigma_i \leftarrow g'^{1/(sk+\gamma^i)}; \quad u_i \leftarrow u^{\gamma^i}; \quad (8)$$

$$\text{acc} \leftarrow \text{acc} \cdot g'_{L+1-i}; \quad V \leftarrow V \cup \{i\}; \quad (9)$$

$$\text{wit}_i \leftarrow \{\sigma_i, u_i, g_i, w, V\}. \quad (10)$$

5. Sends $(A_i, \sigma, c, s'', \text{wit}_i, g_i, g'_i, i)$.

6. Publishes updated V, acc .

Prover finishes:

1. Computes $s \leftarrow s' + s''$.

2. Stores *non-revocation claim* $C_{NR} \leftarrow (A_i, \sigma, c, s, \text{wit}_i, g_i, g'_i, i)$.

TEST Tests

$$\frac{e(g_i, \text{acc}_V)}{e(g, w)} \stackrel{?}{=} z; \quad (11)$$

$$e(pk \cdot g_i, \sigma_i) \stackrel{?}{=} e(g, g'_i); \quad (12)$$

$$e(\sigma, y \cdot \hat{h}^c) \stackrel{?}{=} e(h_0 \cdot h_1^{m_2} h_2^s g_i, \hat{h}). \quad (13)$$

5.3 Revocation

Issuer revokes user with m_2 value that corresponds to accumulator acc , index i , and valid index set V :

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

2. Computes $\text{acc} \leftarrow \text{acc} / g'_{L+1-i}$.

3. Publishes $\{V, \text{acc}\}$.

6 Presentation

Verifier sends nonce n_1 to the Prover.

Prover prepares all claim pairs (C_1, C_2) he wants to submit:

1. Initiates \mathcal{T} and \mathcal{C} as empty sets. Generates random 1024-bit \widetilde{m}_1 .

2. For all claim pairs (C_1, C_2) executes Section 6.1.

3. Executes Section 6.2 once.

4. For all claim pairs (C_1, C_2) executes Section 6.3.

5. Executes Section 6.3 once.

Then Verifier

1. For all claim pairs (C_1, C_2) executes Section 6.5.

2. Executes Section 6.6 once.

6.1 Initial preparation

Let \mathcal{A} be the set of *all* attribute identifiers present, of which \mathcal{A}_r are the identifiers of attributes that are revealed to the Verifier, and $\mathcal{A}_{\bar{r}}$ are those that are hidden.

Non-revocation proof. Verifier starts:

1. Loads Issuer's public revocation key $p = (h, h_1, h_2, \tilde{h}, \hat{h}, u, pk, y)$.

Prover continues:

1. Loads Issuer's public revocation key $p = (h, h_1, h_2, \tilde{h}, \hat{h}, u, pk, y)$.
2. Loads the non-revocation claim $C_{NR} \leftarrow (A_i, \sigma, c, s, \text{wit}_i, g_i, g'_i, i)$;
3. Obtains recent V, acc (from Verifier, Sovrin link, or elsewhere).
4. Updates C_{NR} :

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

$$V_{old} \leftarrow V.$$

Here V_{old} is taken from wit_i and updated there.

5. Selects random $\rho, \rho', r, r', r'', r''', o, o' \bmod q$;
6. Computes

$$E \leftarrow h^\rho \tilde{h}^o \quad D \leftarrow g^r \tilde{h}^{o'}; \quad (14)$$

$$A \leftarrow \sigma \tilde{h}^\rho \quad \mathcal{G} \leftarrow g_i \tilde{h}^r; \quad (15)$$

$$\mathcal{W} \leftarrow w \hat{h}^{r'} \quad \mathcal{S} \leftarrow \sigma_i \hat{h}^{r''} \quad (16)$$

$$\mathcal{U} \leftarrow u_i \hat{h}^{r'''} \quad (17)$$

and adds these values to \mathcal{C} .

7. Computes

$$m \leftarrow \rho \cdot c; \quad t \leftarrow o \cdot c; \quad (18)$$

$$m' \leftarrow r \cdot r''; \quad t' \leftarrow o' \cdot r''; \quad (19)$$

8. Generates random $\tilde{\rho}, \tilde{o}, \tilde{o}', \tilde{c}, \tilde{m}, \tilde{m}', \tilde{t}, \tilde{t}', \tilde{m}_2, \tilde{s}, \tilde{r}, \tilde{r}', \tilde{r}'', \tilde{r}''' \bmod q$.

9. Computes

$$\overline{T}_1 \leftarrow h^{\tilde{\rho}} \tilde{h}^{\tilde{o}} \quad \overline{T}_2 \leftarrow E^{\tilde{c}} h^{-\tilde{m}} \tilde{h}^{-\tilde{t}} \quad (20)$$

$$\overline{T}_3 \leftarrow e(A, \hat{h})^{\tilde{c}} \cdot e(\tilde{h}, \hat{h})^{\tilde{r}} \cdot e(\tilde{h}, y)^{-\tilde{\rho}} \cdot e(\tilde{h}, \hat{h})^{-\tilde{m}} \cdot e(h_1, \hat{h})^{-\tilde{m}_2} \cdot e(h_2, \hat{h})^{-\tilde{s}} \quad (21)$$

$$\overline{T}_4 \leftarrow e(\tilde{h}, \text{acc})^{\tilde{r}} \cdot e(1/g, \hat{h})^{\tilde{r}'} \quad \overline{T}_5 \leftarrow g^{\tilde{r}} \tilde{h}^{\tilde{o}'} \quad (22)$$

$$\overline{T}_6 \leftarrow D^{\tilde{r}''} g^{-\tilde{m}'} \tilde{h}^{-\tilde{t}'} \quad \overline{T}_7 \leftarrow e(pk \cdot \mathcal{G}, \hat{h})^{\tilde{r}''} \cdot e(\tilde{h}, \hat{h})^{-\tilde{m}'} \cdot e(\tilde{h}, \mathcal{S})^{\tilde{r}} \quad (23)$$

$$\overline{T}_8 \leftarrow e(\tilde{h}, u)^{\tilde{r}} \cdot e(1/g, \hat{h})^{\tilde{r}'''} \quad (24)$$

and add these values to \mathcal{T} .

TEST Tests that for

$$\begin{array}{llll} \tilde{\rho} = \rho & \tilde{o} = o & \tilde{o}' = o' & \tilde{c} = c \\ \tilde{m} = m & \tilde{m}' = m' & \tilde{t} = t & \tilde{t}' = t' \\ \tilde{m}_2 = m_2 & \tilde{s} = s & \tilde{r} = r & \tilde{r}' = r' \\ \tilde{r}'' = r'' & \tilde{r}''' = r''' & & \end{array}$$

the following holds:

$$E \stackrel{?}{=} h^{\tilde{\rho}} \tilde{h}^{\tilde{\sigma}} \quad 1 \stackrel{?}{=} E^{\tilde{c}} h^{-\tilde{m}} \tilde{h}^{-\tilde{t}} \quad (25)$$

$$\frac{e(h_0 \mathcal{G}, \hat{h})}{e(A, y)} \stackrel{?}{=} e(A, \hat{h})^{\tilde{c}} \cdot e(\tilde{h}, \hat{h})^{\tilde{r}} \cdot e(\tilde{h}, y)^{-\tilde{\rho}} \cdot e(\tilde{h}, \hat{h})^{-\tilde{m}} \cdot e(h_1, \hat{h})^{-\tilde{m}_2} \cdot e(h_2, \hat{h})^{-\tilde{s}} \quad (26)$$

$$\frac{e(\mathcal{G}, \text{acc})}{e(g, \mathcal{W})^z} \stackrel{?}{=} e(\tilde{h}, \text{acc})^{\tilde{r}} \cdot e(1/g, \hat{h})^{\tilde{r}'} \quad D \stackrel{?}{=} g^{\tilde{r}} \tilde{h}^{\tilde{\sigma}'} \quad (27)$$

$$1 \stackrel{?}{=} D^{\tilde{r}''} g^{-\tilde{m}'} \tilde{h}^{-\tilde{t}'} \quad \frac{e(pk \cdot \mathcal{G}, \mathcal{S})}{e(g, g')} \stackrel{?}{=} e(pk \cdot \mathcal{G}, \hat{h})^{\tilde{r}''} \cdot e(\tilde{h}, \hat{h})^{-\tilde{m}'} \cdot e(\tilde{h}, \mathcal{S})^{\tilde{r}} \quad (28)$$

$$\frac{e(\mathcal{G}, u)}{e(g, \mathcal{U})} \stackrel{?}{=} e(\tilde{h}, u)^{\tilde{r}} \cdot e(1/g, \hat{h})^{\tilde{r}'''} \quad (29)$$

Validity proof

0.1 For each non-revealed attribute $i \in \mathcal{A}_{\bar{r}}$ generate random 592-bit number \tilde{m}_i , except for already generated \tilde{m}_2 and \tilde{m}_1

1. For each credential $C = (\mathcal{I} = \{m_j\}, A, e, v)$ and Issuer's public key pk_I :

1.1 Choose random 2128-bit r ;

1.2 Take n, S from pk_I compute

$$A' \leftarrow AS^r \pmod{n} \text{ and } v' \leftarrow v - e \cdot r \text{ in integers.} \quad (30)$$

Also compute $e' \leftarrow e - 2^{596}$.

2.1 Generate random 456-bit number \tilde{e} and random 3060-bit number \tilde{v} .

2.2 Compute

$$T \leftarrow (A')^{\tilde{e}} \left(\prod_{j \in \mathcal{A}_{\bar{r}} \cap \mathcal{I}} (R_i)^{\tilde{m}_j} \right) (S^{\tilde{v}}) \pmod{n}.$$

2.3 Add T to \mathcal{T} , A' to \mathcal{C} .

2. For each predicate $p: m_j \geq z_j$:

(a) Load Z, S from issuer's public key.

(b) Let $\Delta \leftarrow m_j - z_j$ and find (possibly by exhaustive search) u_1, u_2, u_3, u_4 such that

$$\Delta = (u_1)^2 + (u_2)^2 + (u_3)^2 + (u_4)^2;$$

(c) Generate random 2128-bit numbers $r_1, r_2, r_3, r_4, r_\Delta$, compute

$$T_1 \leftarrow Z^{u_1} S^{r_1} \pmod{n} \quad T_2 \leftarrow Z^{u_2} S^{r_2} \pmod{n} \quad (31)$$

$$T_3 \leftarrow Z^{u_3} S^{r_3} \pmod{n} \quad T_4 \leftarrow Z^{u_4} S^{r_4} \pmod{n} \quad (32)$$

$$T_\Delta \leftarrow Z^\Delta S^{r_\Delta} \pmod{n}; \quad (33)$$

and add these values to \mathcal{C} .

(d) Take \tilde{m}_j generated at step 0.1, generate random 592-bit numbers $\tilde{u}_1, \tilde{u}_2, \tilde{u}_3, \tilde{u}_4$, generate random 672-bit numbers $\tilde{r}_1, \tilde{r}_2, \tilde{r}_3, \tilde{r}_4, \tilde{r}_\Delta$ compute

$$\overline{T}_1 \leftarrow Z^{\tilde{u}_1} S^{\tilde{r}_1} \pmod{n} \quad \overline{T}_2 \leftarrow Z^{\tilde{u}_2} S^{\tilde{r}_2} \pmod{n} \quad (34)$$

$$\overline{T}_3 \leftarrow Z^{\tilde{u}_3} S^{\tilde{r}_3} \pmod{n} \quad \overline{T}_4 \leftarrow Z^{\tilde{u}_4} S^{\tilde{r}_4} \pmod{n} \quad (35)$$

$$\overline{T}_\Delta \leftarrow Z^{\tilde{m}_j} S^{\tilde{r}_\Delta} \pmod{n}; \quad (36)$$

and add this values to \mathcal{T} in the order $\overline{T}_1, \overline{T}_2, \overline{T}_3, \overline{T}_4, \overline{T}_\Delta$.

(e) Generate random 2787-bit number $\tilde{\alpha}$ and compute

$$Q \leftarrow \overline{T}_1^{\tilde{u}_1} \overline{T}_2^{\tilde{u}_2} \overline{T}_3^{\tilde{u}_3} \overline{T}_4^{\tilde{u}_4} S^{\tilde{\alpha}} \pmod{n}$$

Add Q to \mathcal{T} .

6.2 Hashing

Prover computes

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

and sends c_H to Verifier.

6.3 Final preparation

For primary claim C_1 and non-revocation claim C_2 Prover

1. Computes

$$\begin{aligned} \widehat{\rho} &\leftarrow \widetilde{\rho} - c_H \rho \bmod q & \widehat{o} &\leftarrow \widetilde{o} - c_H \cdot o \bmod q \\ \widehat{c} &\leftarrow \widetilde{c} - c_H \cdot c \bmod q & \widehat{o}' &\leftarrow \widetilde{o}' - c_H \cdot o' \bmod q \\ \widehat{m} &\leftarrow \widetilde{m} - c_H m \bmod q & \widehat{m}' &\leftarrow \widetilde{m}' - c_H m' \bmod q \\ \widehat{t} &\leftarrow \widetilde{t} - c_H t \bmod q & \widehat{t}' &\leftarrow \widetilde{t}' - c_H t' \bmod q \\ \widehat{m}_2 &\leftarrow \widetilde{m}_2 - c_H m_2 \bmod q & \widehat{s} &\leftarrow \widetilde{s} - c_H s \bmod q \\ \widehat{r} &\leftarrow \widetilde{r} - c_H r \bmod q & \widehat{r}' &\leftarrow \widetilde{r}' - c_H r' \bmod q \\ \widehat{r}'' &\leftarrow \widetilde{r}'' - c_H r'' \bmod q & \widehat{r}''' &\leftarrow \widetilde{r}''' - c_H r''' \bmod q. \end{aligned}$$

and add them to \mathcal{X} .

4.1 Compute

$$\begin{aligned} \widehat{e} &\leftarrow \widetilde{e} + c_H \cdot e'; \\ \widehat{v} &\leftarrow \widetilde{v} + c_H v'. \end{aligned}$$

4.2 For all $j \in A_{\overline{r}}$ compute

$$\widehat{m}_j \leftarrow \widetilde{m}_j + c_H m_j.$$

The values $Pr_C = (\widehat{e}, \widehat{v}, \widehat{m}_j, A')$ are the *sub-proof* for claim C_1 .

4.3 For each predicate $p: m_j \geq z_j$:

- (a) For $1 \leq i \leq 4$ compute $\widehat{u}_i \leftarrow \widetilde{u}_i + c_H u_i$.
- (b) For $1 \leq i \leq 4$ compute $\widehat{r}_i \leftarrow \widetilde{r}_i + c_H r_i$.
- (c) Compute $\widehat{r}_\Delta \leftarrow \widetilde{r}_\Delta + c_H r_\Delta$.
- (d) Compute $\widehat{\alpha} \leftarrow \widetilde{\alpha} + c_H (r_\Delta - u_1 r_1 - u_2 r_2 - u_3 r_3 - u_4 r_4)$.

The values $Pr_p = (\{\widehat{u}_i\}, \{\widehat{r}_i\}, \widehat{r}_\Delta, \widehat{\alpha}, \widehat{m}_j)$ are the sub-proof for predicate p .

6.4 Sending

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

6.5 Verification

For the claim pair (C_1, C_2) Verifier retrieves relevant variables from $\mathcal{X}, \{Pr_C\}, \{Pr_p\}, \mathcal{C}$.

6.5.1 Non-revocation check

Verifier computes

$$\widehat{T}_1 \leftarrow E^{c_H} \cdot h^{\widehat{\rho}} \cdot \widetilde{h}^{\widehat{o}} \quad \widehat{T}_2 \leftarrow E^{\widehat{c}} \cdot h^{-\widehat{m}} \cdot \widetilde{h}^{-\widehat{t}} \quad (37)$$

$$\widehat{T}_3 \leftarrow \left(\frac{e(h_0 \mathcal{G}, \widehat{h})}{e(A, y)} \right)^{c_H} \cdot e(A, \widehat{h})^{\widehat{c}} \cdot e(\widetilde{h}, \widehat{h})^{\widehat{r}} \cdot e(\widetilde{h}, y)^{-\widehat{\rho}} \cdot e(\widetilde{h}, \widehat{h})^{-\widehat{m}} \cdot e(h_1, \widehat{h})^{-\widehat{m}_2} \cdot e(h_2, \widehat{h})^{-\widehat{s}} \quad (38)$$

$$\widehat{T}_4 \leftarrow \left(\frac{e(\mathcal{G}, \text{acc})}{e(g, \mathcal{W})z} \right)^{c_H} \cdot e(\widetilde{h}, \text{acc})^{\widehat{r}} \cdot e(1/g, \widehat{h})^{\widehat{r}'} \quad \widehat{T}_5 \leftarrow D^{c_H} \cdot g^{\widehat{r}} \widetilde{h}^{\widehat{o}'} \quad (39)$$

$$\widehat{T}_6 \leftarrow D^{\widehat{r}''} \cdot g^{-\widehat{m}'} \widetilde{h}^{-\widehat{t}'} \quad \widehat{T}_7 \leftarrow \left(\frac{e(pk \cdot \mathcal{G}, \mathcal{S})}{e(g, g')} \right)^{c_H} \cdot e(pk \cdot \mathcal{G}, \widehat{h})^{\widehat{r}''} \cdot e(\widetilde{h}, \widehat{h})^{-\widehat{m}'} \cdot e(\widetilde{h}, \mathcal{S})^{\widehat{r}} \quad (40)$$

$$\widehat{T}_8 \leftarrow \left(\frac{e(\mathcal{G}, u)}{e(g, \mathcal{U})} \right)^{c_H} \cdot e(\widetilde{h}, u)^{\widehat{r}} \cdot e(1/g, \widehat{h})^{\widehat{r}'''} \quad (41)$$

and adds these values to \widehat{T} .

6.5.2 Validity

Verifier uses all Issuer public key pk_I involved into the credential generation and the received $(c, \widehat{e}, \widehat{v}, \{\widehat{m}_i\}, A')$. He also uses revealed m_i for $i \in A_r$. He initiates $\widehat{\mathcal{T}}$ as empty set.

1. For each credential C consider the attributes I used in C and take sub-proof $(\widehat{e}, \widehat{v}, A')$. Then compute

$$\widehat{T} \leftarrow \left(\frac{Z}{\left(\prod_{i \in A_r \cap I} (R_i)^{m_i} \right) (A')^{2^{596}}} \right)^{-c} (A')^{\widehat{e}} \left(\prod_{i \in A_{\overline{r}} \cap I} (R_i)^{\widehat{m}_i} \right) S^{\widehat{v}} \pmod{n}. \quad (42)$$

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

2. For each predicate $p : m_j \geq z_j$
 - (a) Using Pr_p and \mathcal{C} compute

$$\widehat{T}_i \leftarrow T_i^{-c} Z^{\widehat{u}_i} S^{\widehat{r}_i} \pmod{n} \quad \text{for } 1 \leq i \leq 4; \quad (43)$$

$$\widehat{T}_\Delta \leftarrow (T_\Delta Z^{z_j})^{-c} Z^{\widehat{m}_j} S^{\widehat{r}_\Delta} \pmod{n}; \quad (44)$$

$$\widehat{Q} \leftarrow (T_\Delta)^{-c} T_1^{\widehat{u}_1} T_2^{\widehat{u}_2} T_3^{\widehat{u}_3} T_4^{\widehat{u}_4} S^{\widehat{\alpha}} \pmod{n}, \quad (45)$$

and add these values to $\widehat{\mathcal{T}}$ in the order $\widehat{T}_1, \widehat{T}_2, \widehat{T}_3, \widehat{T}_4, \widehat{T}_\Delta$.

6.6 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.

6.7 Implementation notice

The exponentiation, modulo, and inverse operations are implemented in the Charm library for the class *integer*.

6.8 Why it works

6.8.1 Signature proof

Suppose that the Prover submitted the right values. Then Equation (42) can be viewed as

$$\widehat{T} = Z^{-c} \left(\prod_{i \in A_r} (R_i)^{cm_i} \right) (A')^{\widehat{e} + c \cdot e' + c 2^{596}} \left(\prod_{i \in A_{\overline{r}}} (R_i)^{\widehat{m}_i + cm_i} \right) S^{\widehat{v} + cv'}. \quad (46)$$

If we reorder the multiples, we get

$$\widehat{T} = Z^{-c} \left(\prod_{i \in A_r} (R_i)^{cm_i} \right) \left(\prod_{i \in A_{\overline{r}}} (R_i)^{cm_i} \right) (A')^{c \cdot (e' + 2^{596})} S^{cv'} \left(\prod_{i \in A_{\overline{r}}} (R_i)^{\widehat{m}_i} \right) (A')^{\widehat{e}} S^{\widehat{v}} \quad (47)$$

The last three factors multiple to T so we get

$$\widehat{T} = \left(\frac{Z}{(A')^e S^{v'} \prod_i (R_i)^{m_i}} \right)^{-c} T \quad (48)$$

From Equation (30) we obtain that $(A')^e = A^e S^{v-v'}$, so we finally get

$$\widehat{T} = \left(\frac{Z}{A^e S^v \prod_i (R_i)^{m_i}} \right)^{-c} T \quad (49)$$

From the definition of A we get that

$$A^e S^v = \frac{Z}{\prod_i (R_i)^{m_i}}, \quad (50)$$

which implies

$$\widehat{T} = T.$$

6.8.2 Predicate proof

For the proof to be verified, the values $\widehat{T}_i, \widehat{Q}$ derived in Equations (44),(43),(45) must coincide with \overline{T}_i, Q computed by Prover.

We have

$$\widehat{T}_\Delta = (T_\Delta Z^{z_j})^{-c} Z^{\widehat{m}_j} S^{\widehat{r}_\Delta} = Z^{-c\Delta - cz_j + \widehat{m}_j} S^{-cr_\Delta + \widehat{r}_\Delta} = Z^{\widehat{m}_j} S^{\widehat{r}_\Delta} = \overline{T}_\Delta. \quad (51)$$

We also have

$$\widehat{T}_i = T_i^{-c} Z^{\widehat{u}_i} S^{\widehat{r}_i} = Z^{-cu_i + \widehat{u}_i} S^{-cr_i + \widehat{r}_i} = Z^{\widehat{u}_i} S^{\widehat{r}_i} = \overline{T}_i. \quad (52)$$

Finally,

$$\begin{aligned} \widehat{Q} &= (T_\Delta)^{-c} T_1^{\widehat{u}_1} T_2^{\widehat{u}_2} T_3^{\widehat{u}_3} T_4^{\widehat{u}_4} S^{\widehat{\alpha}} = \\ &= Z^{-c\Delta} S^{-cr_\Delta} \left(T_1^{\widehat{u}_1} T_2^{\widehat{u}_2} T_3^{\widehat{u}_3} T_4^{\widehat{u}_4} \right) (T_1^{cu_1} T_2^{cu_2} T_3^{cu_3} T_4^{cu_4}) S^{\widehat{\alpha} + c(r_\Delta - u_1 r_1 - u_2 r_2 - u_3 r_3 - u_4 r_4)} = \text{Equation (45)} = \\ &= Z^{-c\Delta} S^{-cr_\Delta} Q(T_1^{cu_1} T_2^{cu_2} T_3^{cu_3} T_4^{cu_4}) S^{c(r_\Delta - u_1 r_1 - u_2 r_2 - u_3 r_3 - u_4 r_4)} = \\ &= Q Z^{-c\Delta + cu_1^2 + cu_2^2 + cu_3^2 + cu_4^2} S^{-cr_\Delta + r_1 cu_1 + r_2 cu_2 + r_3 cu_3 + r_4 cu_4 + c(r_\Delta - u_1 r_1 - u_2 r_2 - u_3 r_3 - u_4 r_4)} = Q. \end{aligned} \quad (53)$$

7 Changelog

7.1 7 Feb 2018 (version 0.3)

Type-3-pairing-based revocation added.

7.2 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.

7.3 13 July 2017

Added:

- Proof of correctness for Issuer's setup in Section 4.2;
- Verification of correctness of setup: steps 1.0.1, 1.0.2;
- Proof of correctness for Prover's blinded attributes: steps 1.3.1, 1.3.2, 1.4;
- Verification Prover'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.