Anonymous credentials 2.0

Michael Lodder, Dmitry Khovratovich

23 January 2019, version 0.1

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 *verifier*. 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, BBS+[2]-based credentials with external proofs are described. *External proofs* are proofs of statements about attribute values, that are performed

[1]Technically it is possible to support credentials with different l_a, but in Sovrin for simplicity it is set $l_a = 256$.
[2]See section 4.3-5.1

in other proof systems and are described in other documents. Examples are range proofs and AuthZ proofs using the Bulletproofs framework. For this, Holder creates separate commitments, one per attribute value, and includes the proof of correctness into the BBS+ proof of signature knowledge. Holder then presents external proofs using these commitments.

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

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 sets the attribute he controls, including the credential index (used for non-revocation) and returns credential C to holder. Issuer adds the index to the accumulator of issued credentials.
5. Holder approaches verifier. Verifier sends the Proof Request $\mathcal{E}$ to holder. The Proof Request contains the credential schema $\mathcal{S}_E$, disclosure predicates $\mathcal{D}$, and AuthZ specification $\mathcal{A}$. The predicates for attribute m and value V can be of form $m < V$ 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 or another storage.
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 credential schema $\mathcal{S}$ with l attributes $m_1, m_2, \ldots, m_l$ and the set of hidden attributes $D_P \subset \{1, 2, \ldots, l\}$. In Sovrin, m_1 is reserved for the link secret of the holder, m_2 is reserved for the credential index, m_3 is reserved for the policy address I. By default, $1 \in D_P$ whereas $2, 3 \notin D_P$. Let us denote $D_I = \{1, 2, \ldots, l\} \setminus D_P$.

4.2 Credential Cryptographic Setup

In Sovrin, issuers use BBS+ [?] for primary credentials, although other signature types will be supported too. The BBS+ issuer chooses a pairing friendly curve. Sovrin uses BLS12-381.

1. Issuer sets up a pairing operation on BLS12-381:
$$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T,$$
where groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ have prime order p.
2. For group $\mathbb{G}_1$ he prepares:
    - Generators $\widehat{g_1}, h_0, h_1, \ldots, h_L$.
3. For group $\mathbb{G}_2$ he prepares:
    - Generator g_2.
    - Random integer $x < p$.

- Computes $w \leftarrow g_2^x$.

Issuer's public key for schema $\mathcal{S}$ is

$$pk_I = \{\widehat{g_1}, h_0, h_1, \dots, h_L, g_2, w\}.$$

Issuer's secret key is x.

Therefore, $\mathcal{S}$ consists of the following elements:

$$\mathcal{S} = \{\Phi, D_I, pk_I\}$$

4.2.1 Proof of Setup Correctness

Issuer also creates the proof of knowledge of the secret key x (if needed):

1. $\overline{g_1}$ is random element of $\mathbb{G}_1$.
2. $\overline{g_2} \leftarrow \overline{g_1}^x$;
3. r is a random integer $< p$.
4.

$$\begin{aligned} t_1 \leftarrow & g_2^r;\ t_2 \leftarrow \overline{g_1}^r; \\ c \leftarrow & H(\overline{g_1}||\overline{g_2}||\widehat{g_1}||g_2||t_1||t_2); \\ s \leftarrow & r - cx; \end{aligned}$$

Proof $\pi \leftarrow (\overline{g_1}, \overline{g_2}, t_1, t_2, s, c)$. To verify the proof, Holder computes

$$t_1' \leftarrow g_2^s w^c;\ t_2' \leftarrow \overline{g_1}^s \overline{g_2}^c; \tag{1}$$

and checks if

$$c = H(\overline{g_1}||\overline{g_2}||\widehat{g_1}||g_2||t_1'||t_2')$$

4.3 Non-revocation Cryptographic Setup

In AnonCreds 2.0 issuers use Merkle trees to track revocation status of credentials. Each credential is given an index from 1 to $L = 2^q$, and indices of non-revoked credentials are the Merkle tree leafs.

Let V denote the set of non-revoked indices, and H be the Merkle tree hash function.

Then we define

$$H_{[i]} = \begin{cases} H(i);\ \text{if } i \in V \\ 0,\ \text{if } i \notin V \end{cases} \tag{2}$$

$$H_{[2i,2i+1]} = H(H_{[2i]}, H_{[2i+1]}); \qquad H_{[2^k i, 2^k i + 2^k - 1]} = H(H_{[2^k i, 2^k i + 2^{k-1} - 1]}, H_{[2^k i + 2^k, 2^k i + 2^k - 1]}). \tag{3}$$

$$\mathcal{T} = H_{[1,L]}. \tag{4}$$

Let us denote by $+i$ the addition of non-revoked index i to the tree, and by $-i$ the removal of index i from the tree $\mathcal{T}$.

5 Issuance of Credentials

5.1 Holder Setup

Holder:

- Loads credential schema $\mathcal{S}$.
- Sets hidden attributes $\{m_i\}_{i \in D_P}$.
- Establishes a connection with issuer and gets 128-bit nonce n_0 either from issuer or as a precomputed value. Holder is known to issuer with identifier $\mathcal{H}$.

Holder prepares data for the credential:

1. Generate random

- $s' \mod q$
- $\widetilde{s'} \mod q$
- $\{r_i\}_{i \in D_P}$

2. Compute taking $h_0, \{h_i\}$ from P_k:

$$U \leftarrow (h_0^{s'}) \prod_{i \in D_P} h_i^{m_i} \tag{5}$$

3. Compute

$$\widetilde{U} \leftarrow (h_0^{\widetilde{s'}}) \prod_{i \in D_P} h_i^{r_i}; \qquad c \leftarrow H(U || \widetilde{U} || n_0); \tag{6}$$

$$\widehat{s'} \leftarrow \widetilde{s'} - cs'; \qquad \{\widehat{r_i} \leftarrow r_i - cm_i\}_{i \in D_P} \tag{7}$$

4. Send $\{U, c, \widehat{s'}, \{\widehat{r_i}\}_{i \in D_P}\}$ to the issuer.

5.2 Credential Issuance

1. Issuer receives U.
2. Issuer verifies the correctness of holder's input:
   (a) Compute $\widehat{U} \leftarrow (U^c)(h_0^{\widehat{s'}}) \prod_{i \in D_I} h_i^{\widehat{r_i}}$
   (b) Verify $c = H(U || \widehat{U} || n_0)$
3. Issuer determines the attributes he controls: $\{m_i\}_{i \in D_I}$. Let the credential index be i_0, i.e. $m_2 = i_0$.
4. Issuer generates random $e, s'' \mod q$.
5. Issuer computes

$$B \leftarrow \widehat{g_1} U h_0^{s''} \prod_{i \in D_I} h_i^{m_i}; \qquad A \leftarrow B^{\frac{1}{e+x}}; \tag{8}$$

$$V \leftarrow V \cup \{i_0\}; \qquad \mathcal{T} \leftarrow \mathcal{T} + i_0. \tag{9}$$

6. Issuer returns $(A, e, s'', \{m_i\}_{i \in D_I})$.
7. Issuer publishes updated $V, \mathcal{T}$ on the ledger.
8. Prover computes

$$B \leftarrow \widehat{g_1} g_{i_0} U h_0^{s''} \prod_{i \in D_I} h_i^{m_i}; \tag{10}$$

9. Prover verifies

$$e(A, wg_2^e) \overset{?}{=} e(B, g_2). \tag{11}$$

10. Prover stores credential $\mathcal{C} = \{\{m_i\}_{1 \le i \le L}, A, e, (s' + s'') \bmod q\}$.

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 $\mathcal{T} \leftarrow \mathcal{T} - i$.
3. Publish $\{V, \mathcal{T}\}$.

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 $X - x_1$ attributes are unknown (denote them A_h), which form $x_4 = (X - x_1 - x_2)$ 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. Let D_r denote the set of indices for which a predicate proof is computed.

The proof request also specifies A_h, ϕ, A_v and the set $\mathcal{D}$ of predicates:

- For range proof $m_i \in [a; b]$ – predicates $a \leq m_i < 2^n$ and $m_i \leq b \leq 2^n$ for some $n < 255$.
- For non-revocation proof specify Merkle tree $\mathcal{T}$ and request the predicate $m_2 \in \mathcal{T}$.
- For proof of membership of attribute m_i in some Merkle tree $\mathcal{T}'$ request the predicate $m_i \in \mathcal{T}'$.
- For proof of membership of attribute m_i in set $\mathcal{S}$ request the predicate $m_i \in \mathcal{S}$.

. Along with a proof request, Verifier also generates and sends 80-bit nonce n_1. The predicate proof is computed separately from the credential correctness proof by using the commitments to m_i values generated in the latter.

7.2 Proof Preparation

Holder prepares for each credential:

1. Generate random $\{\widetilde{m_i}\}_{i \notin D}$ of length 256 bits (group size).
2. Run Algorithm 1 with inputs $D, \mathcal{C}, \{\widetilde{m_i}\}_{i \notin D}, D_r$. The outputs $\mathcal{R}, \mathcal{T}$ are added to $\mathcal{R}_{full}, \mathcal{T}_{full}$, and outputs $\mathcal{U}, \mathcal{S}_r$ is stored.
3. Compute
$$c_H \leftarrow H(\mathcal{R}_{full}, \mathcal{T}_{full}, n_0).$$
4. For each $j \notin D$ compute $\widehat{m}_j \leftarrow \widetilde{m_j} + c_H m_j \bmod p$. and add $\{\widehat{m_i}\}_{i \notin D}$ to $\mathcal{P}_{full}$.
5. Run Algorithm 2 with inputs $c_H, e, \mathcal{U}$ and add output $\mathcal{P}$ to $\mathcal{P}_{full}$.
6. Send $\{c_H, \mathcal{R}_{full}, \mathcal{P}_{full}, \{m_i\}_{i \in D}\}$ to Verifier.
7. For each predicate $2^n > m_i \geq v_i$ run Algorithms 4,5 on the input $\{n, s'_i, Z'_i = Z_i h_1^{-v_i}\}$. The output proof π is sent to Verifier.
8. For each predicate $2^n > v_i \geq m_i$ run Algorithms 4,5 on the input $\{n, s'_i, Z'_i = h_1^{v_i}/Z_i\}$. The output proof π is sent to Verifier.
9. For each Merkle set membership predicate $m_i \in \mathcal{T}$ run Algorithm **??** on input $(i, Z_i, \mathcal{T})$. The output proof π is sent to Verifier.
10. For each linear set membership predicate $m_i \in \mathcal{S}$ run Algorithms **??** on input $(i, Z_i, \mathcal{S})$. The output proof π is sent to Verifier.

7.3 Verification

For the credential C

1. Run Algorithm 3 with inputs $\mathcal{R}, \mathcal{P}, \{m_i\}_{i \in D}, \{\widehat{m_i}\}_{i \notin D}$. Add output $\widehat{\mathcal{T}}$ to $\widehat{\mathcal{T}}_{full}$.
2. Check that
$$c_H = H(\mathcal{R}_{full}, \widehat{\mathcal{T}}_{full}, n_0)$$
3. Check that
$$e(A', w) = e(\overline{A}, g_2) \tag{12}$$

4. For every predicate $2^n > m_j \geq v_j$ or $m_j \leq v_j < 2^n$ verify the Bulletproof π_j by Algorithm 6 with input $\{n, Z_j, \pi_j\}$.
5. For each Merkle set membership predicate $m_i \in \mathcal{T}$ verify the Bulletproof proof π by Algorithm **??** on input $(i, Z_i, \mathcal{T}, \pi)$.
6. For each linear set membership predicate $m_i \in \mathcal{S}$ verify the Bulletproof proof π by Algorithm **??** on input $(i, Z_i, \mathcal{S}, \pi)$.

8 Algorithms

Algorithm 1 Selective disclosure: commitment step

```
Input: $D, \mathcal{C} = \{\{m_i\}_{1\leq i\leq L}, A, e, s\}, \{\widetilde{m_i}\}_{i\notin D}, D_r$
1. Generate random $r_1, r_2, \widetilde{e}, \widetilde{r_2}, \widetilde{r_3}, \widetilde{s'}$;
2. For commitments, generate random $\{s'_i\}_{i\in D_r}$ and compute
       $Z_i \leftarrow h_1^{m_i} h_0^{s'_i},\ i \in D_r.$
3. Compute
   • $B \leftarrow \widehat{g}_1 h_0^s \prod_{1\leq i\leq L} h_i^{m_i}$
   • $A' \leftarrow A^{r_1}$
   • $\overline{A} \leftarrow A'^{-e} B^{r_1}$
   • $d \leftarrow B^{r_1} h_0^{r_2}$
   • $r_3 \leftarrow r_1^{-1} modp$
   • $s' \leftarrow s - r_2 r_3$
   • $t_1 \leftarrow A'^{-\widetilde{e}} h_0^{\widetilde{r_2}}$
   • $t_2 \leftarrow d^{\widetilde{r_3}} h_0^{-\widetilde{s'}} \prod_{i\notin D} h_i^{-\widetilde{m_i}}$
   • $R \leftarrow \widehat{g}_1 \prod_{i\in D} h_i^{m_i}$
   • $t'_i \leftarrow h_1^{\widetilde{m_i}} h_0^{\widetilde{s'_i}},\ i \in D_r.$
Output: $\mathcal{R} = \{A', \overline{A}, d, R, \{Z_i\}_{i\in D_r}\}$, $\mathcal{T} = \{t_1, t_2, r_1, r_2, r_3, s'\}$, $\mathcal{U} = \{\widetilde{e}, \widetilde{r_2}, \widetilde{r_3}, \widetilde{s'}, \{\widetilde{s'_i}\}_{i\in D_r}\}, \mathcal{S}_r = \{s'_i\}$.
```

Algorithm 2 Selective disclosure: opening step

```
Input: $c_H, e, r_1, r_2, r_3, s', \{\widetilde{m_i}\}_{i\notin D}, \mathcal{U} = \{\widetilde{e}, \widetilde{r_2}, \widetilde{r_3}, \widetilde{s'}\}$.
1. Compute
   $\widehat{e} \leftarrow \widetilde{e} - c_H e;\quad \widehat{r_2} \leftarrow \widetilde{r_2} - c_H r_2;\quad \widehat{r_3} \leftarrow \widetilde{r_3} - c_H r_3;\quad \widehat{s'} \leftarrow \widetilde{s'} - c_H s';$
   $\{\widehat{s'_i} \leftarrow \widetilde{s'_i} - c_H s'_i\}_{i\in D_r};$
Output: $\mathcal{R} = \{A', \overline{A}, d, R\}$, $\mathcal{P} = \{\widehat{e}, \widehat{r_2}, \widehat{r_3}, \widehat{s'}, \{\widehat{s'_i}\}_{i\in D_r}\}$.
```

9 Changelog

9.1 9 Feb 2018 (version 0.4)

Formatting and updates for committed attributes

```
Algorithm 3 Selective disclosure: Verification step
Input: $c_H, \mathcal{R} = \{A', \overline{A}, d, R, \{Z_i\}\}$, $\mathcal{P} = \{\widehat{e}, \widehat{r_2}, \widehat{r_3}, \widehat{s'}, \{\widehat{s'_i}\}_{i \in D_r}\}, \{m_i\}_{i \in D}, \{\widehat{m_i}\}_{i \notin D}$
    1. Check that $A' \neq 1$;
    2. Compute
        $\widehat{t_1} \leftarrow (\overline{A}/d)^{c_H} A'^{-\widehat{e}} h_0^{\widehat{r_2}};$   (13)
        $\widehat{t_2} \leftarrow R^{c_H} d^{\widehat{r_3}} h_0^{-\widehat{s'}} \prod_{i \notin D} h_i^{-\widehat{m_i}}.$   (14)
        $\widehat{t'_i} \leftarrow Z_i^{c_H} h_0^{\widehat{s'_i}} h_1^{\widehat{m_i}}, \ i \in D_r.$   (15)
Output: $\widehat{\mathcal{T}} = \{\widehat{t_1}, \widehat{t_2}, \{\widehat{t'_i}\}_{i \in D_r}\}$.
```

```
Algorithm 4 Range Bulletproofs: commitment step
Input: $m_j, z_j$
    1. Prover takes preselected generators $g, h$, generator vectors. See Section 4.
    2. Generate random value $r_1 \mod p$
    3. Calculate $\Delta$ such that:
        $\Delta \leftarrow \begin{cases} z_j - m_j; & \text{if } * \equiv \leq \\ z_j - m_j - 1; & \text{if } * \equiv < \\ m_j - z_j; & \text{if } * \equiv \geq \\ m_j - z_j - 1; & \text{if } * \equiv > \end{cases}$
    4. Compute
        $V \leftarrow g^{\Delta} h^{r_1};$   (16)
                                              (17)
Output:$V, r_1, \Delta$
```

```
Algorithm 5 Range Bulletproofs: challenge step
Input: $V, r_1, \Delta$
Proofs are generated as described in Section 4.1. We use the inner product argument as described in Section 4.2 to reduce proof size.
    • $V, r_1, \Delta$ are inputs to bulletproofs black box
Output: $\pi$
```

```
Algorithm 6 Range Bulletproofs: verification step
Input: $\pi$
Proofs are verified as described in Section 4.1.
    1. Verify($\pi$)
Output: accept if successful or $\perp$ on failure
```

Algorithm 7 Linear set memberships: commitment step

```
Input: $m_j$, set $\mathcal{S}$ of length $n$
  1. Create bit vector $\mathbf{v} \in \{0,1\}^n$ with the index of $m_j$ in set $\mathcal{S}$ to 1 and all others to 0.
  2. Generate random values $\{r_i \mod p\}_{1 \le i \le n}$
  3. Generate random value $r_j \mod p$
  4. Compute commitment vector using bullet proof generators $\mathbf{g}, \mathbf{h}$: $\mathbf{a} \leftarrow g_i^{a_i} h_i^{r_i}$
  5. Compute commitment for value using bulletproof value generators: $V \leftarrow g^{m_j} h^{r_j}$
Output: $V, \mathbf{a}, \{r_i\}$
```

9.2 7 Feb 2018 (version 0.3)

Type-3-pairing-based revocation added.

9.3 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.

9.4 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.