Anonymous credentials

Dmitry Khovratovich

May 13, 2016

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 $m_1, m_2, \ldots, m_l$. 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 2.2 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.

2.1 Common parameters

Some parameters are common for all users and are generated as follows.

1. Generate random 256-bit prime ρ and a random 1376-bit number b such that $\Gamma = b\rho + 1$ is prime and ρ does not divide b;
2. Generate random $g' < \Gamma$ such that $g'^b \neq 1 \pmod{\Gamma}$ and compute $g = g'^b \neq 1$.
3. Generate random $r < \rho$ and compute $h = g^r$.

Then (Γ, ρ, g, h) are public parameters.

2.2 Issuer setup

Let l be the number of attributes we work with. Let P be a description of the attribute set (types, number, length). Every credential is bind to a pseudonym, which is derived from the master secret m_1.

Steps to set up the issuer

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 residue[1] S modulo n;
3. Select random $x_Z, x_{R_1}, \ldots, x_{R_l} \in [2; p'q' - 1]$ 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 $pk_I = (n, S, Z, R_1, R_2, \ldots, R_l, P)$ and the private key is $sk_I = (p, q)$.

2.3 Pseudonym registration

For the master secret m_1 and Issuer's public key pk_I Prover

1. Generates random $r < \rho$ and computes $\text{nym} = g^{m_1} h^r$.
2. Sends nym to the Issuer.

2.4 Issuance

Let $m_2, m_3, .., m_l$ be integer attribute values, all 256-bit, known to the Issuer.

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.

1.4 Prover sends U to the Issuer.

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

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

2.2 Issuer computes

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

and

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

2.3 Issuer sends (A, e, v'') to the Prover.

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

Prover stores credential $(\{m_i\}, A, e, v)$.

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

2.5 Proof 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 $\mathcal{A}_{\overline{r}}$ are those that are hidden.

0.0 Verifier sends nonce n_1 to the Prover.

0.1 For each non-revealed attribute $i \in \mathcal{A}_{\overline{r}}$ generate random 592-bit number $\widetilde{m_i}$.

1. For each credential $C = (\mathcal{I} = \{m_i\}, A, e, v)$ and Issuer's public key pk_I compute values T and A'

   1.1 Choose random 2128-bit r;

   1.2 (*signature randomization*) Take n, S from pk_I compute

$$A' \leftarrow AS^r \pmod{n} \text{ and } v' \leftarrow v - e \cdot r \text{ in integers.} \tag{1}$$

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

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

   2.2 Compute

$$T \leftarrow (A')^{\widetilde{e}} \left(\prod_{i \in A_{\overline{r}} \cap \mathcal{I}} (R_i)^{\widetilde{m_i}} \right) (S^{\widetilde{v}}) \pmod{n}.$$

3 Gather all computed T and A' into sets $\{T\}$ and $\{A\}$ respectively. Compute

$$c \leftarrow H(\{A'\}, \{T\}, n_1)$$

4.1 For each credential $C = (\mathcal{I} = \{m_i\}, A, e, v)$ and Issuer's public key pk_I compute

$$\widehat{e} \leftarrow \widetilde{e} + c \cdot e';$$
$$\widehat{v} \leftarrow \widetilde{v} + cv'.$$

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

$$\widehat{m}_i \leftarrow \widetilde{m_i} + cm_i.$$

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

Then $(c, \{\widehat{e}\}, \{\widehat{v}\}, \{\widehat{m}_i\}, \{A'\})$ is a proof sent to the Verifier.

2.6 Verification

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 $i \in A_r$.

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

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

2. Gather computed $\widehat{T}$ into the set $\{\widehat{T}\}$ and compute

$$\widehat{c} \leftarrow H(\{A'\}, \{\widehat{T}\}, n_1);$$

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

2.7 Implementation notice

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

2.8 Why it works

Suppose that the Prover submitted the right values. Then Equation (2) can be viewed as

$$\widehat{T} = Z^{-c} \left(\prod_{i \in A_r} (R_i)^{cm_i} \right) (A')^{\widetilde{e}+c\cdot e'+c2^{596}} \left(\prod_{i \in A_{\overline{r}}} (R_i)^{\widetilde{m_i}+cm_i} \right) S^{\widetilde{v}+cv'}. \tag{3}$$

If we reorder the multiples, we get

$$\widehat{T} = Z^{-c} \left(\prod_{i \in A_r} (R_i)^{cm_i} \right) \left(\prod_{i \in A_{\overline{r}}} (R_i)^{cm_i} \right) (A')^{c\cdot(e'+2^{596})} S^{cv'} \left(\prod_{i \in A_{\overline{r}}} (R_i)^{\widetilde{m_i}} \right) (A')^{\widetilde{e}} S^{\widetilde{v}} \tag{4}$$

The last three factors multiple to T so we get

$$\widehat{T} = \left(\frac{Z}{(A')^e S^{v'} \prod_i (R_i)^{m_i}} \right)^{-c} T \tag{5}$$

From Equation (1) we obtain that $(A')^e = A^e S^{v-v'}$, so we finally get

$$\widehat{T} = \left(\frac{Z}{A^e S^v \prod_i (R_i)^{m_i}} \right)^{-c} T \tag{6}$$

From the definition of A we get that

$$A^e S^v = \frac{Z}{\prod_i (R_i)^{m_i}}, \tag{7}$$

which implies

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

A Additional math

Prover is a person with certain identity X, consisting of several attributes $m_1, m_2, \ldots, m_l$. The attributes must be integers or convertible to them. The integers have bit length around 256 bits, so longer attributes must be encoded, hashed, or partitioned to fit the length restriction.

The statements about attribute m_i use *commits*. For public fixed parameters g, h Prover selects random r and computes commit $C \leftarrow g^{m_i} h^r$. The value m_i is then called a committed value. He is then able to prove various assertions about the committed value, including the existence of Issuer's signature on it.