Anonymous credentials

Dmitry Khovratovich

May 10, 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 N. It is supposed that Prover has been registered as N at the Issuer, and communicated (part of) his identity X to him. After that the Issuer can issue a credential that couples N 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 N_V is required, it can be generated from a master secret S_U together with N in a way that N can not be linked to N_V. However, Prover can prove that the credential he presents was issued to a pseudonym derived from the same master secret as used to produce N_V.

2 Simple example

2.1 Issuer setup

Let l be the number of attributes we work with. No pseudonyms are used, so no master secret. Let P be a description of the attribute set (types, number, length).

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 $1 \leq i \leq l$.

(similarly to *keygen* from `https://github.com/JHUISI/charm/blob/dev/charm/schemes/pksig/pksig_cl03.py`)

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

2.2 Issuance

Let $m_1, m_2, .., m_l$ be integer attribute values, all 256-bit.

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

1.2 Prover computes $U \leftarrow S^{v'} \pmod n$ 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

$$Q \leftarrow \frac{Z}{US^{v''}(R_1^{m_1} R_2^{m_2} \cdots R_l^{m_l})} \pmod n.$$

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

2.3 Proof preparation

Let $\mathcal{A}_r$ be the indices of attributes that are revealed to the Verifier, and $\mathcal{A}_{\overline{r}}$ be those that are hidden.

Let S be part of the Issuer public key, and $(\{m_i\}, A, e, v)$ be the credential.

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

1.1 Choose random 2128-bit r_A;

1.2 (*signature randomization*) Using S compute

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

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

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

3 Compute

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

4 Compute

$$\begin{aligned} \widehat{e} &\leftarrow \widetilde{e} + c \cdot e'; \\ \widehat{v} &\leftarrow \widetilde{v} + cv'; \\ \widehat{m}_i &\leftarrow \widetilde{m_i} + cm_i \quad \text{for all } i \in A_{\overline{r}}. \end{aligned}$$

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

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

2.4 Verification

Verifier uses Issuer's public key $(n, S, Z, R_1, R_2, \ldots, R_l, P)$ and the received $(c, \widehat{e}, \widehat{v}, \{\widehat{m}_i\}, A')$.

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

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

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

2.5 Implementation notice

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

2.6 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.

A.1 Proof that committed value is smaller than a certain number

In a simplified example, a Prover wants to prove that Y is a k-bit number i.e. $Y = y_1 y_2 \ldots y_k < 2^k$ in his commitment $C = g^Y h^R \pmod{n}$. Prover selects a large prime q and arbitrary $r_1, r2, \ldots, r_k$ such that

$$R = r_1 \cdot 2^{k-1} + r_2 \cdot 2^{k-2} + \cdots + r_{k-1} 2 + r_k \pmod{q}.$$

Then he computes $C_i \leftarrow g^{y_i} h^{r_i}$ for $1 \le i \le k$ and sends all C_i to the Verifier, who checks that $(..((C_1)2C_2)2 \cdots)2C_k = C$. Finally, for each i Prover proves that he knows x_i and r_i. The latter proof reduces to the proof that Prover knows the discrete logarithm r_i in either h^{r_i} or gh^{r_i} (the Prover does not disclose in which one).