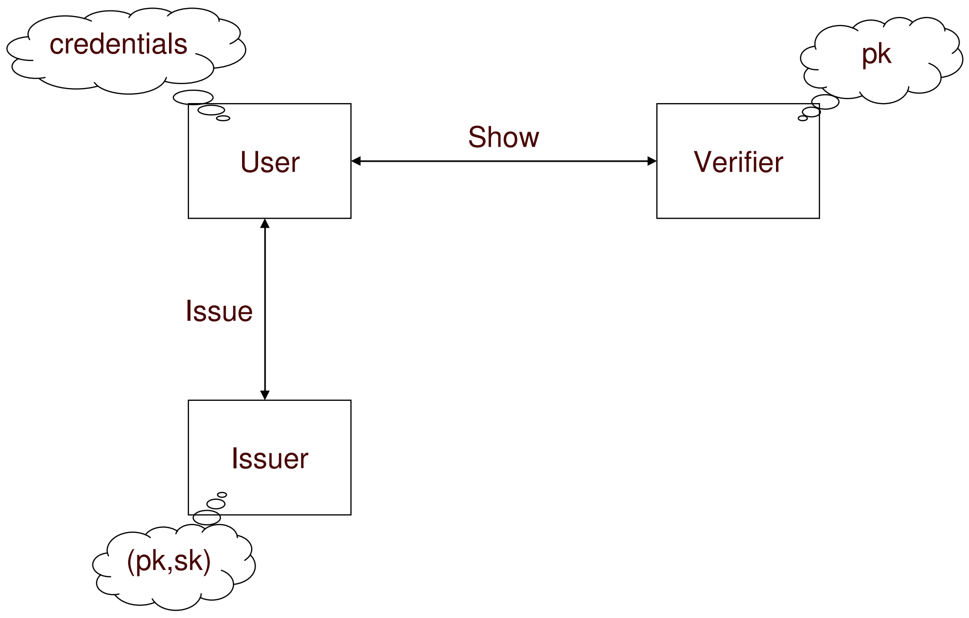
# A Quick Introduction to Anonymous Credentials

Gregory Neven
IBM Zürich Research Laboratory
August 2008

This document is intended as a quick introduction to anonymous credentials for a technically, but not cryptographically trained audience. We avoid all mathematical and cryptographic detail here; rather, we try to sketch a black-box API that captures the main functionalities and security features of anonymous credentials, allowing the reader to quickly gain an intuition into what anonymous credentials can and cannot do.

## 1. Communication model



In the typical "wine-shop" use case, the User is the customer, the Verifier is the merchant, and the Issuer is the government, or any other instance that is trusted to issue age certificates.

The User engages in an Issue protocol with an Issuer to obtain a valid credential on a certain set of attributes. The credential is valid under the Issuer's public key *pk*, of which only the Issuer knows the corresponding secret key *sk*. In its simplest form, the Issue protocol is a two-round interaction where the User submits a request containing her attributes, and the Issuer certifies the fact that the User has the claimed attributes by returning the credential; more generally, the credentials can be generated through a multi-round interaction.The User convinces a verifier that she has a certain set of attributes by engaging in a Show protocol. Again, this can be a simple two-round protocol or a more complex interaction.

## 2. Anonymous credentials and selective showing of attributes

Intuitively, an anonymous credential can be thought of as a digital signature by the Issuer on a list of attribute-value pairs, e.g. the list

```
(fname="Alice", lname="Anderson", bdate="1977/05/10", nation="DE").
```

The most straightforward way for the User to convince a Verifier of her list of attributes would be to simply transmit her credential to the Verifier. This approach has a number of disadvantages, most notably

- that the User has to reveal *all* of her attributes so that the Verifier can check the signature;
- that the Verifier can reuse the credential to impersonate Alice wrt other Verifiers.

With anonymous credentials, the User never transmits the credential itself, but rather uses it to convince the Verifier that her attributes satisfy certain properties – without leaking anything about the credential other than the shown properties. This has the obvious advantage that the Verifier can no longer reuse the credential to impersonate Alice. Another advantage is that anonymous credentials allow the User to reveal a selected subset of her attributes. In the example above, Alice could for example reveal her first name and last name, but keep her birth date hidden.

Stronger even, apart from showing the exact value of an attribute, the User can even convince the Verifier that some complex predicate over the attributes holds. In particular, she can demonstrate any predicate consisting of the following operations:

- **Integer arithmetic:** $att+c$ , $att+att$ , $att \cdot c$ : sum of attributes and/or constants, product of attributes with constants
- **Comparative:** $exp_1 = exp_2$ , $exp_1 < exp_2$ , $exp_1 > exp_2$ , $exp_1 \leq exp_2$ , $exp_1 \geq exp_2$ : compare arithmetic expressions over attributes/constants
- **Logical:** $pred_1$ *AND* $pred_2$ , $pred_1$ *OR* $pred_2$

Note that revealing a subset of attributes comes down to the special of showing that the predicate

$$att_1 = val_1 \;\; AND \;\; att_2 = val_2 \;\; AND \;\; \ldots$$

holds. In the example above, Alice could show that she's an overage national of an EU country by showing that he has a credential satisfying

$$\texttt{bdate} \leq today - 18y \;\; AND \;\; (\, \texttt{nation="DE"} \; OR \; \texttt{nation="FR"} \; OR \ldots ).$$

The Show protocol does not leak any information other than the truth of the statement. So in the example above, not only does Alice's name remain hidden from the Verifier, but so do her exact birth date and nationality; the only information leaked is that her birth date is more than 18 years ago, and that her nationality is one of the EU countries.

The Show protocol also allows to prove statements about multiple credentials. For example, when Alice's passport and credit card are anonymous credentials issued by her government and her bank, respectively, then she can show to a Verifier that she has a valid passport certifying that she is over 18 years of age and a valid credit card issued to the same name:

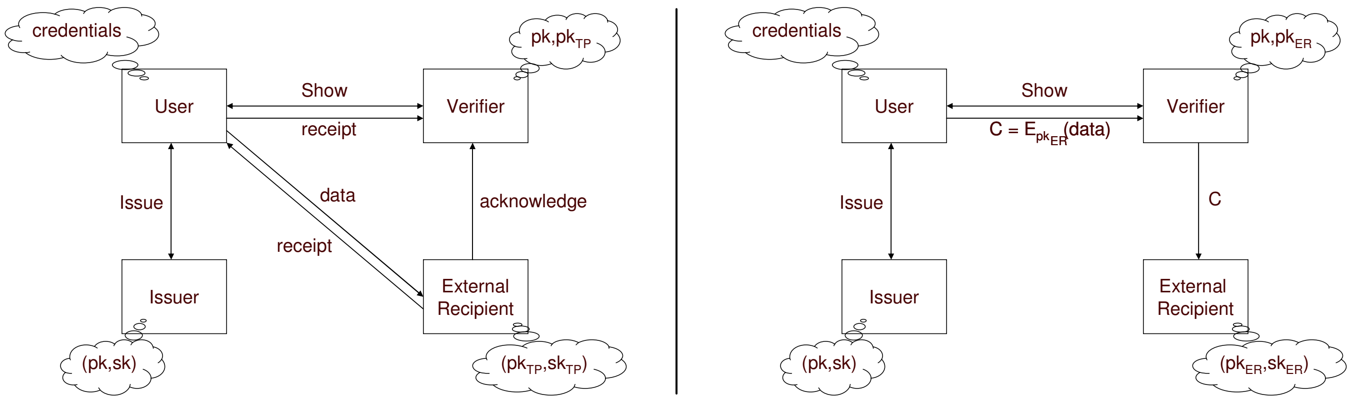
$$\begin{gathered}\texttt{pass.bdate} \leq today - 18y \;\; AND \;\; \texttt{pass.fname} = \texttt{ccard.fname} \\ AND \;\; \texttt{pass.lname} = \texttt{ccard.lname}\,.\end{gathered}$$

## 3. Verifiable encryption

There are situations where multiple parties are involved in the same transaction, and each of these parties need different information from the User. For example, in the wine shop use case, the merchant needs to know whether Alice is overage, and the shipping company needs to know her address. The merchant should not learn her address, but wants to be sure that she gives her real address to the shipping company, and not that of an underage friend.

Another example is that the merchant may require Alice to submit her real identity to a trusted third party (TTP), so that in case of dispute or fraud the TTP can be contacted to reveal Alice's identity. The merchant wants to be sure that Alice submitted her real identity to the TTP, and not some bogus identity. The TTP in this case plays the role of an identity escrow agent.

More generally, there are situations where the Verifier wants to be sure that the User submitted a certain piece of personal data to an External Recipient, but the content of these data should remain hidden from the Verifier. A "low-tech" solution to this problem is to have the User send this data directly to the External Recipient, who either returns a receipt to the User that the User can show to the Verifier, or who contacts the Verifier directly to acknowledge that it received the correct data. This situation is depicted in the left picture below.



A more "high-tech" setting is depicted in the right picture. Here, the User sends the required data to the Verifier, but encrypted under the public key of the External Recipient. When she uses a *verifiable* encryption scheme, the User can prove predicates about the content of the encrypted data of the same form as those in the Show protocol to the Verifier, to assure the Verifier that the ciphertext C actually contains the correct data. (Without such a proof, the User could easily encrypt bogus data, since the Verifier does not know the decryption key $sk_{ER}$ anyway.) The External Recipient does not need to be online at the time of the transaction; rather, the Verifier can forward the ciphertext only when it is actually needed, e.g. when the item is being shipped, or when a dispute actually occurs.

## 4. Limited spending

### 4.1 One-time spending

Because of the anonymity of anonymous credentials, a Verifier cannot tell whether two successful logins were initiated by the same user or by different users. This is good for privacy, but becomes a problem in situations where one wants to place a bound on the number of times that a single credential can be used. We first sketch a simple solution for the special case where a credential can only be used (or "spent") once, and then present a more advanced solution that works for arbitrary bounds.

When a credential is only to be used once (a so-called "one-time credential"), the issuer includes an additional attribute `credentialID` in the credential that is set to a random value. To log into the system, the User has to reveal the value of `credentialID` to the Verifier. The Verifier simply maintains the list of `credentialID`s that it has seen, and denies access when the revealed `credentialID` already occurs in the list.

This solution works fine as long as there is only a single Verifier, or as long as all Verifiers have access to a common database that keeps track of spent `credentialID`s. It is impossible to prevent a User from overspending a one-time credential by using it at multiple Verifiers that are not in constant communication with each other, but there are ways to detect such behavior after the fact. Namely, as part of the showing protocol, the Verifier can choose a random challenge *chall*, which the User has to respond to with a value *resp* that computed as a function of *chall* and some attribute that uniquely identifies the User, e.g. `name`, together with a proof that *resp* was correctly computed. The protocol is such that a single challenge-response pair (*chall*, *resp*) does not reveal anything about `name`, but `name` is easily computable from two pairs (*chall, resp*) and (*chall', resp'*) whenever $chall \neq chall'$ (which happens with overwhelming probability if the challenges are long enough). By storing triples of the form (`credentialID`, *chall*, *resp*) for each login, Verifiers can later on compare their lists to detect overspent credentials and reveal the identity of fraudulent users.

## 4.2 Multi-time spending

The above technique is easily extended to allow spending up to *n* times as follows: include *n* different random attributes `credentialID1`,…,`credentialIDn` in the credential, and let the User reveal one of them each time she logs in. This quickly becomes inefficient however, as the number of attributes in the credential grows linearly with the spending bound. Also, the Issuer has to fix a single bound once and for all; it would be nice if different Verifiers could impose their own bounds on-the-fly.

A more powerful approach is the following. As an extra attribute, each credential includes a random hash key `HK` for a keyed hash function H. The hash function is such that it allows the User to prove statements of the form

$$H_{\mathtt{HK}}(somestring) = somevalue$$

as part of the showing protocol, whereby the hash input and output are revealed, but `HK` remains hidden. When the User logs in for the *i*-th time, she reveals $H_{\mathtt{HK}}(i)$ and proves that the value was correctly computed from `HK` as included in her credential. The Verifier checks that $1 \leq i \leq n$ and that the hash value does not yet occur in its database.

Since the hash function takes any bit string as input, different web sites can easily impose their own spending limits by collecting hash values of the form

$$H_{\mathtt{HK}}(websiteID \parallel i\ )$$

from users. The system is also easily extended to interactive challenge-response protocols that allow after-the-fact detection of abuse.