Anonymous Credits

Jonathan Katz* Samuel Schlesinger*

Abstract

We describe a scheme that can be used for anonymous payments.

1 Definitions

In the model we consider, there is a single *issuer* and multiple clients. The issuer issues credits to clients, who can redeem those credits with the issuer in an anonymous fashion. (Of course, one can allow for multiple issuers who operate independently. However, we assume that credits issues by a given issuer can only be redeemed at that issuer.) Formally:

Definition 1. *An* anonymous credit scheme *(ACS) consists of algorithms/protocols* (KeyGen, Issue, Spend) *with the following syntax:*

- KeyGen *is run by an issuer to generate keys* $(\mathsf{pk}, \mathsf{sk})$.
- Issue *is an interactive protocol run by the issuer and a client. The issuer has as input its private key* sk *and a (non-negative integer) value* c*, and the client has as input the issuer's public key* pk *and the same value* c*. At the end of the protocol, the client outputs a credit token* (tok, c) *with* $\mathsf{tok} = \perp$ *indicating that it detected cheating by the issuer.*
- Spend *is an interactive protocol run by the issuer and a client. The issuer has as input its private key* sk *and a (non-negative integer) value* s*, and the client has as input the issuer's public key* pk*, the same value* s*, and a token* (tok, c) *with* $s \leq c$*. At the end of the protocol, the issuer outputs a nullifier* k *with* $k = \perp$ *indicating that it detected cheating by the client; the client outputs* (tok', c') *with* $\mathsf{tok} = \perp$ *indicating that it detected cheating by the issuer.*

Correctness requires that (in honest executions) issuance always succeeds in producing a token worth the appropriate number of credits, and that spending s credits from token worth c credits should succeed (at the issuer) and result in a new token worth $c' = c - s$ credits for the client. We also require that the nullifiers output by the issuer are distinct.

Definition 2. *An anonymous credit scheme is* correct *if for any efficient adversary* $\mathcal{A}$ *the probability that* flag *is set to 1 in the following experiment is negligible:*

1. *Generate keys* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}$*, and give* pk *to* $\mathcal{A}$*. Set* $\mathsf{flag} := 0$.
2. $\mathcal{A}$ *may interact with oracles as follows. On its* i*th oracle query:*

*Google.

- *If the ith query is of the form* $\mathsf{Issue}(c)$*, run* Issue *honestly between the issuer and a client on shared input* c*; let* (tok_i, c_i) *be the client's output. If* $\mathsf{tok}_i = \perp$ *or* $c_i \neq c$*, set* $\mathsf{flag} := 1$*.*
- *If the ith query is of the form* $\mathsf{Spend}(s, j)$ *we require* $j < i$ *and* $s \leq c_j$*. Run* Spend *honestly between the issuer and a client on shared input* s *and with the client using token* (tok_j, c_j)*. Let* k *be the output of the issuer, and let* (tok_i, c_i) *be the output of the client. Set* $\mathsf{flag} := 1$ *if* $k = \perp$ *or* $\mathsf{tok}_i = \perp$ *or* $c_i \neq c_j - s$ *or* k *was every previously output be the issuer following a* Spend *query.*

For security, we require *fiscal soundness* (to protect against malicious clients) and *anonymity* (to protect against a malicious issuer). Formal definitions are omitted from the present draft.

2 Construction

2.1 Overview

Before providing the details of the construction, we give a high-level overview. Our ACS relies on (privately verifiable) BBS "signatures." Roughly, a credit token for c credits is a signature on (c, k), where k is a random nullifier chosen by the client. That is, issuance of a token for c credits is done by having a client choose a uniform nullifier k and then obtain a BBS signature from the issuer on (c, k), without revealing k. Spending s credits using a c-credit token (with $c \geq s$) associated with nullifier k then involves the following steps:

1. The client sends k to the issuer, and proves possession of a BBS signature on (c, k) with $c \geq s$. (We currently use binary decomposition to prove the latter; other methods could also be used.)
2. The client obtains a fresh BBS signature from the issuer on $(c - s, k^*)$ for a fresh k^*. (Note this must be done even when $c - s = 0$ if the client wants to maintain anonymity.)

The purpose of the nullifier is to prevent double spending: the issuer can store a database of previously used nullifiers, and reject a spend attempt if k has been used before. This check is not included in the description below.

2.2 Technical Details

We now provide the details of our scheme. We remark that, in contrast to the informal description in the previous section, our scheme actually uses BBS signatures on vectors of three attributes, with one of the attributes serving as a randomizer to prevent linkability.

Setup. Fix a group $\mathbb{G}$ of prime order q with generator g, along with (random) generators $h_1, h_2, h_3 \in \mathbb{G}$. (These could be chosen by the issuer, or fixed public values.) Finally, we assume the number of credits a token can contain (and hence the maximum possible spend amount) is strictly less than 2^ℓ for known ℓ.[1]

Key generation. An issuer chooses a secret key $x \leftarrow \mathbb{Z}_q$; the associated public key is $w := g^x$.

Token issuance. To issue a token for c credits to some client, the client and issuer run the following protocol:

[1] While one can consider a non-power-of-two bound, it is not clear that there is any advantage to doing so here.

1. The client chooses $k, r \leftarrow \mathbb{Z}_q$, sets $K := h_2^k h_3^r$, and computes a non-interactive proof of knowledge of k, r as follows:

   (a) Choose $k', r' \leftarrow \mathbb{Z}_q$ and set $K_1 := h_2^{k'} h_3^{r'}$.

   (b) Compute $\gamma := H(K \| K_1)$; then set $\bar{k} := \gamma \cdot k + k'$ and $\bar{r} := \gamma \cdot r + r'$.

   The client sends $K, \gamma, \bar{k}, \bar{r}$ to the issuer.

2. The issuer then does:

   - (**Verification.**) Compute $K_1 := h_2^{\bar{k}} h_3^{\bar{r}} K^{-\gamma}$ and check that $H(K \| K_1) \stackrel{?}{=} \gamma$.
   - (**Issuance.**) The issuer then chooses $e \leftarrow \mathbb{Z}_q$ and sends $(A, e) = \left((g \cdot h_1^c \cdot K)^{1/(e+x)}, e \right)$ to the client. It also proves that it computed this value correctly by proving that $\log_A (g \cdot h_1^c \cdot K) = \log_g (g^e \cdot w)$ using a standard "equality-of-discrete-logarithms" proof. That is, let $X_A = g \cdot h_1^c \cdot K$ and $X_g = g^e \cdot w$. The issuer does:

     (a) Choose $\alpha \leftarrow \mathbb{Z}_q$ and compute $Y_A := A^\alpha$ and $Y_g := g^\alpha$.

     (b) Compute $\gamma := H(A \| e \| Y_A \| Y_g)$.

     (c) Compute $z := \gamma \cdot (x + e) + \alpha$, and send γ, z to the client.

3. The client verifies the proof γ, z by computing $Y'_A := A^z \cdot X_A^{-\gamma}$ and $Y'_g := g^z \cdot X_g^{-\gamma}$ and then checking if $H(A \| e \| Y'_A \| Y'_g) \stackrel{?}{=} \gamma$. If so, the client outputs $((A, e, k, r), c)$; otherwise, the client outputs $\perp$.

Spending. To spend $s < 2^\ell$ credits using a token (A, e, k, r) for c credits (with $s \leq c < 2^\ell$), the client and issuer do the following ($\mathsf{to\_send}_1, \mathsf{to\_send}_2$, and $\mathsf{to\_hash}$ are initially empty):

- (**Client spend.**) The client reveals k and proves knowledge of a BBS signature on a vector of the form (c, k, r) for some c, r with $c \geq s$. This is done as follows:

  1. The client chooses $r_1, r_2, c', r', e', r'_2, r'_3 \leftarrow \mathbb{Z}_q$. It then sets $B := g \cdot h_1^c h_2^k h_3^r$, $A' := A^{r_1 r_2}$, $\bar{B} := B^{r_1}$, $r_3 := r_1^{-1}$, $A_1 := (A')^{e'} \cdot \bar{B}^{r'_2}$, $A_2 := \bar{B}^{r'_3} \cdot h_1^{c'} h_3^{r'}$, $\mathsf{to\_send}_1 := \mathsf{to\_send}_1 \| k \| A' \| \bar{B}$, and $\mathsf{to\_hash} := \mathsf{to\_hash} \| A_1 \| A_2$.

  2. Let $i_{\ell-1}, \ldots, i_0 \in \{0, 1\}$ be the binary representation of $c - s$. The client chooses $k^*, s_0, \ldots, s_{\ell-1} \leftarrow \mathbb{Z}_q$ and sets $\mathsf{com}_0 := h_1^{i_0} h_2^{k^*} h_3^{s_0}$ and $\mathsf{com}_j := h_1^{i_j} h_3^{s_j}$ for $j = 1, \ldots, \ell - 1$; it then sets $\mathsf{to\_send}_1 := \mathsf{to\_send}_1 \| \mathsf{com}_0 \| \cdots \| \mathsf{com}_{\ell-1}$. It then does:

     (a) Set $C_{0,0} := \mathsf{com}_0$ and $C_{0,1} := \mathsf{com}_0 / h_1$.

     (b) Choose $k'_0, s'_0, \gamma_0, w_0, z_0 \leftarrow \mathbb{Z}_q$.

     (c) If $i_0 = 0$ set $C'_{0,0} := h_2^{k'_0} h_3^{s'_0}$ and $C'_{0,1} := h_2^{w_0} h_3^{z_0} C_{0,1}^{-\gamma_0}$.
     If $i_0 = 1$ set $C'_{0,0} := h_2^{w_0} h_3^{z_0} C_{0,0}^{-\gamma_0}$ and $C'_{0,1} := h_2^{k'_0} h_3^{s'_0}$.

     (d) Set $\mathsf{to\_hash} := \mathsf{to\_hash} \| C'_{0,0} \| C'_{0,1}$.

     Next, for $j = 1, \ldots, \ell - 1$ the client does:

     (a) Set $C_{j,0} := \mathsf{com}_j$ and $C_{j,1} := \mathsf{com}_j / h_1$.

     (b) Choose $s'_j, \gamma_j, z_j \leftarrow \mathbb{Z}_q$.

(c) If $i_j = 0$ set $C'_{j,0} := h_3^{s'_j}$ and $C'_{j,1} := h_3^{z_j} C_{j,1}^{-\gamma_j}$.
If $i_j = 1$ set $C'_{j,0} := h_3^{z_j} C_{j,0}^{-\gamma_j}$ and $C'_{j,1} := h_3^{s'_j}$.
(d) Set $\mathsf{to\_hash} := \mathsf{to\_hash} \| C'_{j,0} \| C'_{j,1}$.

Finally, it sets $r^* := \sum_{j=0}^{\ell-1} 2^j s_j$.

3. The client chooses $k', s' \leftarrow \mathbb{Z}_q$ and sets $C := h_1^{-c'} h_2^{k'} h_3^{s'}$ and $\mathsf{to\_hash} := \mathsf{to\_hash} \| C$. Then it computes $\gamma := H(\mathsf{to\_send}_1 \| \mathsf{to\_hash}) \in \mathbb{Z}_q$.
4. The client computes $\bar{e} := -\gamma \cdot e + e'$, $\bar{r}_2 := \gamma r_2 + r'_2$, $\bar{r}_3 := \gamma r_3 + r'_3$, $\bar{c} := -\gamma c + c'$, $\bar{r} := -\gamma r + r'$, and $\mathsf{to\_send}_2 := \mathsf{to\_send}_2 \| \bar{e} \| \bar{r}_2 \| \bar{r}_3 \| \bar{c} \| \bar{r}$. Then
   (a) If $i_0 = 0$ set $\gamma_{0,0} := \gamma - \gamma_0$, $w_{0,0} := \gamma_{0,0} \cdot k^* + k'_0$, $w_{0,1} := w_0$, $z_{0,0} := \gamma_{0,0} \cdot s_0 + s'_0$, and $z_{0,1} := z_0$.
   If $i_0 = 1$ set $\gamma_{0,0} := \gamma_0$, $w_{0,0} := w_0$, $w_{0,1} := (\gamma - \gamma_{0,0}) \cdot k^* + k'_0$, $z_{0,0} := z_0$, and $z_{0,1} := (\gamma - \gamma_{0,0}) \cdot s_0 + s'_0$.
   (b) Set $\mathsf{to\_send}_2 := \mathsf{to\_send}_2 \| w_{0,0} \| w_{0,1} \| \gamma_{0,0} \| z_{0,0} \| z_{0,1}$.

   Then for $j = 1, \ldots, \ell - 1$ it does:

   (a) If $i_j = 0$ set $\gamma_{j,0} := \gamma - \gamma_j$, $z_{j,0} := \gamma_{j,0} \cdot s_j + s'_j$, and $z_{j,1} := z_j$.
   If $i_j = 1$ set $\gamma_{j,0} := \gamma_j$, $z_{j,0} := z_j$, and $z_{j,1} := (\gamma - \gamma_{j,0}) \cdot s_j + s'_j$.
   (b) Set $\mathsf{to\_send}_2 := \mathsf{to\_send}_2 \| \gamma_{j,0} \| z_{j,0} \| z_{j,1}$.
5. The client sets $\bar{k} := \gamma k^* + k'$, $\bar{s} := \gamma r^* + s'$ and $\mathsf{to\_send}_2 := \mathsf{to\_send}_2 \| \bar{k} \| \bar{s}$. Finally, the client sends $\mathsf{to\_send}_1, \gamma, \mathsf{to\_send}_2$ to the issuer. The client stores $c - s, k^*, r^*$ to be used later below.

- (**Verification by the issuer.**) The issuer, with secret key x, parses $\mathsf{to\_send}_1, \mathsf{to\_send}_2$ as

$$\mathsf{to\_send}_1 = k \| A' \| \bar{B} \| \mathsf{com}_0 \| \cdots \| \mathsf{com}_{\ell-1}$$
$$\mathsf{to\_send}_2 = \bar{e} \| \bar{r}_2 \| \bar{r}_3 \| \bar{c} \| \bar{r} \| w_{0,0} \| w_{0,1} \| \gamma_{0,0} \| z_{0,0} \| z_{0,1} \| \cdots \| \gamma_{\ell-1,0} \| z_{\ell-1,0} \| z_{\ell-1,1} \| \bar{k} \| \bar{s} \ .$$

It rejects if $A' = 1$. Otherwise, it does the following ($\mathsf{to\_hash}$ is initially empty):

1. Set $\bar{A} := (A')^x$, $H_1 := g \cdot h_2^k$, $A_1 := (A')^{\bar{e}} \bar{B}^{\bar{r}_2} \bar{A}^{-\gamma}$, $A_2 := \bar{B}^{\bar{r}_3} h_1^{\bar{c}} h_3^{\bar{r}} H_1^{-\gamma}$, and $\mathsf{to\_hash} := \mathsf{to\_hash} \| A_1 \| A_2$.
2. Do:
   (a) Set $\gamma_{0,1} := \gamma - \gamma_{0,0}$, $C_{0,0} := \mathsf{com}_0$, and $C_{0,1} := \mathsf{com}_0 / h_1$.
   (b) Set $C'_{0,0} := h_2^{w_{0,0}} h_3^{z_{0,0}} C_{0,0}^{-\gamma_{0,0}}$ and $C'_{0,1} := h_2^{w_{0,1}} h_3^{z_{0,1}} C_{0,1}^{-\gamma_{0,1}}$.
   (c) Set $\mathsf{to\_hash} := \mathsf{to\_hash} \| C'_{0,0} \| C'_{0,1}$.

   Then for $j = 1, \ldots, \ell - 1$ do:

   (a) Set $\gamma_{j,1} := \gamma - \gamma_{j,0}$, $C_{j,0} := \mathsf{com}_j$, and $C_{j,1} := \mathsf{com}_j / h_1$.
   (b) Set $C'_{j,0} := h_3^{z_{j,0}} C_{j,0}^{-\gamma_{j,0}}$ and $C'_{j,1} := h_3^{z_{j,1}} C_{j,1}^{-\gamma_{j,1}}$.
   (c) Set $\mathsf{to\_hash} := \mathsf{to\_hash} \| C'_{j,0} \| C'_{j,1}$.
3. The issuer computes $K' := \prod_{i=0}^{\ell-1} \mathsf{com}_i^{2^i}$, $\mathsf{com} := h_1^s \cdot K'$, $C := h_1^{-\bar{c}} h_2^{\bar{k}} h_3^{\bar{s}} \mathsf{com}^{-\gamma}$, and $\mathsf{to\_hash} := \mathsf{to\_hash} \| C$. It aborts if $\gamma \neq H(\mathsf{to\_send}_1 \| \mathsf{to\_hash})$.

- (**Issuing a refund.**) Let K' be the value computed in the previous step. The issuer does the following (note this is almost identical to the issuance protocol described earlier): Choose $e \leftarrow \mathbb{Z}_q$ and send $(A, e) = \left((g \cdot K')^{1/(e+x)}, e\right)$ to the client. Also prove that it computed this value correctly by proving that $\log_A (g \cdot K') = \log_g (g^e \cdot w)$ using a standard "equality-of-discrete-logarithms" proof. That is, let $X_A = g \cdot K'$ and $X_g = g^e \cdot w$. The issuer does:

  1. Choose $\alpha \leftarrow \mathbb{Z}_q$ and compute $Y_A := A^\alpha$ and $Y_g := g^\alpha$.
  2. Compute $\gamma := H(A \| e \| Y_A \| Y_g)$.
  3. Compute $z := \gamma \cdot (x + e) + \alpha$, and send γ, z to the client.

- (**Client computes refund.**) The client verifies the proof γ, z by computing $Y'_A := A^z \cdot X_A^{-\gamma}$ and $Y'_g := g^z \cdot X_g^{-\gamma}$ and then checking if $H(A \| e \| Y'_A \| Y'_g) \stackrel{?}{=} \gamma$. If so, the client outputs $((A, e, k^*, r^*), c - s)$; otherwise, the client outputs $\perp$.