Traceable Anonymous Cash

Traceable anonymous cash sounds like an oxymoron. But the secret of good cryptography is to reveal only the information necessary, and only when appropriate. In this case, the cash is anonymous and untraceable *unless* someone cheats. If they do, the identity of the cheater is revealed (with high probability).

We use secret sharing to do this, but in a new way. We only need the trivial version of secret-sharing in this case (not Shamir's method). Each secret is split into two pieces such that both are needed to reconstruct the secret. For example if $M < n$ is a secret, pick a random $r \in \mathbb{Z}_n$ and set $M_1 = r$, $M_2 = M - r (\mathrm{mod}\ n)$ as the two shares. Then both shares are random numbers but $M_1 + M_2 = M (\mathrm{mod}\ n)$.

Here is the procedure for the spender (the Owner last time) to create a traceable anonymous identity:

1. The spender makes m copies of his/her identity and splits each one in two secret-shared pieces. That is, the copies are $\{Id_1, \ldots, Id_m\}$ and $\{Id'_1, \ldots, Id'_m\}$. The spender's identity will be computable from any pair $Id = Id_i + Id'_i (\mathrm{mod}\ n)$ but from no other combination.

2. The spender chooses $2m$ private (e.g. DES) keys, $K_1, \ldots, K_m, K'_1, \ldots, K'_m$ at random. Key K_i is used to encrypt identity Id_i, and key K'_i is used to encrypt identity Id'_i. Let $f()$ be a suitable encryption function like DES. Each result would look like $f(Id_i G, K_i)$ or $f(Id'_i G, K'_i)$, where f is an encryption function, and G is a "recognizable" string like the name of the bank. It is needed because Id_i is itself just a random string, and decrypting it with a false key would be hard to detect. The keys K_i and K'_i are bit-committed (e.g. by hashing) but kept secret. That is, the public part of the identity info consists of

$$(f(Id_1 G, K_1), \ldots, f(Id_m G, K_m), f(Id'_1 G, K'_1), \ldots, f(Id'_m G, K'_m)$$
$$H(K_1), \ldots, H(K_m), H(K'_1), \ldots, H(K'_m))$$

   while the spender keeps hold of the secret keys:

$$K_1, \ldots, K_m, K'_1, \ldots, K'_m$$

The process above generates one copy of the spender's identity. For the cash protocol below, we will need n copies of the identity info, so we repeat the above process n times. All the random keys K_i are different for each instance of the identity info. Now to create a spendable note, the spender does the following:

1. A spender requesting a unit of cash creates n samples containing:

$$M_i(Bank'sName, Amount, SerialNumber_i, IdentityInfo_i)$$

where the serial number is unique to each bill, and the identity info is computed separately for each note using the protocol above. The spender picks n blinding factors $b_1, \ldots, b_n$ at random and uses these to blind the notes, and presents them to the bank. That is, we send to the bank $b_i^e H(M_i)$ for $i = 1, \ldots, n$ (and bit commitments $H(M_i, b_i)$ for the notes and factors so the spender can't cheat)

2. The bank selects one unit (say the j^{th}) at random and sets it aside. Then it asks the spender to unblind the other $n-1$. The spender provides the blinding factors *and* all the keys K_i and K_i' for those notes. The bank checks the Bank name, amount, and serial number as before.

   A new step is to check the identity info for each note. The bank checks that all the keys K_i and K_i' match their bit commitments in the Idinfo for that note. Then it uses them to look inside the Idinfo. For each note, the bank decrypts all the identity halves Id_i and Id_i', and checks that $Id = Id_i + Id_i' (\mathrm{mod}\ \ n)$ for $i = 1, \ldots, m$.

3. The bank signs the unit $b_j^e H(M_j)$ by computing $b_j H^d(M_j)$ and returns this to the spender.

4. The spender unblinds this unit by multiplying by b_j^{-1} and now has $(M_j, H^d(M))$ which is the public part of the note. Spender also has the secret keys $K_1, \ldots, K_m, K_1', \ldots, K_m'$ for the note M_j. The bank has never seen these keys, nor the note M_j itself.

Spending

Now when the spender takes this note to a merchant, an interactive transaction takes place. In order for the merchant to accept the note, the merchant probes the IdentityInfo in the note. If the spender is honest, this probing is a benign (zero-knowledge) process. The spender will expose only one half of each IdInfo, which reveals nothing about the spender. Here is the protocol:

1. Customer (spender) arrives at the store with the bill. Merchant tosses a coin m times and tells the customer the outcomes.

2. for the i^{th} coin toss, the customer reveals Id_i if the toss was heads, or Id_i' if it was tails by revealing the encryption key K_i or K_i'. The merchant checks the hash of that key, and then uses it to get one half of each identity. But that reveals nothing about the customer's identity.

3. The merchant checks the banks signature and forwards the bill along with the coin flips and the keys K_i or K_i' that were revealed, to the bank.

The customer can't avoid revealing half of each identity for each coin toss. First of all, the identity pieces are almost certainly real. The bank checked the other $n-1$ notes before it signed

this one, including checking that the identity halves matched. It is very unlikely the customer could have cheated when they created the note and got away with it.

The customer can't cheat at step 2 by using a false K_i or K_i' because the keys were bit committed when the note was created (and before the bank signed it). The string G confirms that the decryption worked.

Suppose the customer tries to spend the same note again. Another merchant will toss a coin m times. The customer must reveal m identity halves, and the merchant will forward these (actually their keys) to the same bank. Unless the customer is extremely lucky, the sequence of coin tosses by the second merchant will be different from the first merchant. Suppose toss i is different between merchant 1 and merchant 2. That means the customer revealed both Id_i and Id_i'. When the bank receives those pieces, it will be able to recover the customer's identity. It will also know that the customer deliberately tried to spend twice.

What if the merchant tried to cheat? If they just copy the information that they got from the customer and go to the bank with it, the bank will suspect the merchant. It is very unlikely that a real transaction would happen where the merchant came up with exactly the same sequence of coin tosses. Can the bank forge some identity keys? There is really no way to do this. The keys K_i are all bit committed using a hash function, and the result is signed by the bank. So the merchant can't fake other keys with the same hash value.

The risk to the customer is that the keys K_i must be stored in some kind of computer. If that is stolen, a criminal could try to spend the money twice and only the identity of the original owner would be revealed. In this respect, it is like real money.

The other risk is that you could lose the data and then the money is gone!. In this case, unlike real money, you can keep multiple copies of each note in different digital "wallets" for safety. So long as you manage to track serial numbers and delete notes that have been spent, you will be OK.