Off-Line Cash Transfer by Smart Cards*

Stefan Brands
Centre for Mathematics and Computer Science
P.O. Box 4079
1009 AB Amsterdam
The Netherlands.
E-mail: brands@cwi.nl

September 27, 1994

Abstract

An off-line electronic cash system is presented that offers appreciably greater security and better privacy than currently considered electronic cash systems with similar functionality.

A tamper-resistant smart card, issued by the bank, controls a counter that represents the amount of electronic cash carried by the user. The use of a counter ensures that the computation and communication complexity for paying an amount are independent of the specific amount due, and that conversions between multiple currencies can be made at payment time. Smart cards can transfer electronic cash to POS terminals that need not be physically secured by the bank, without needing on-line verification.

To ensure privacy of payments, the user can insert his smart card into a user-controlled computer, such as a palm top computer or a personal computer, which acts as an intermediary between the smart card and the other party involved in the transaction. Cryptographic software in the user-controlled computer ensures that payments are information-theoretically untraceable and unlinkable.

To pay any specified amount, only 125.5 bytes of data must be transferred, and no on-line computation is required. The dynamic storage requirements per payment can be compressed to a mere 26.5 bytes for the user-controlled computer, and virtually none for the smart card. The smart card can be a smart card capable of performing the well-known Schnorr signature scheme; minor additions to the smart-card code suffice to suit the cash system requirements.

Assuming that the tamper-resistance of the smart cards cannot be broken, the system is provably as hard to break as the Schnorr signature scheme. A build-in mechanism for traceability of double-spent transaction data, which is as hard to break as the blinded Schnorr signature scheme, ensures that the cost of breaking a smart card in practice will significantly exceed the expected financial profit that the attacker can make from this.

*Patent pending.

1 Introduction

From the point of view of security and privacy in privacy-protecting off-line electronic cash systems, coin systems can offer the greatest conceivable level of security. In an appropriately designed coin system, an attacker that manages to break the tamper-resistance of his personal device (and extract its contents), still cannot make a profit out of this without being identified after the fact by the bank. "New" money can be introduced by an attacker, without being noticed, only if he can break the cryptographic security assumptions underlying the system. If the cryptographic assumptions are sufficiently plausible, then it can be expected to be much harder to break the system by a cryptographic attack than to break the tamper-resistance of a personal device.

A practical disadvantage of coin systems is that payments can frequently only be made up by multiple electronic coins. This is certainly a great concern for most of the off-line coin system proposals in the literature, given the large computational resources these systems need to process one electronic coin (due to cut-and-choose in the withdrawal protocol). Although the efficiency of the privacy-protecting off-line coin system that I recently proposed [2] overcomes this problem to a great extent, it might still be worthwhile to look for alternatives, even if these cause a reduction in security.

The most widely applied technique in the design of electronic payment systems is to represent the amount of money carried by the user by a counter in a secured device that has been issued to him. At withdrawal time, the device increases its counter by a specified withdrawal amount, and at payment time it decreases its counter by a specified amount that is transferred to the payee. This ensures that the computation and communication requirements of the parties involved in a transaction are independent of the specific amount that is transferred. An additional advantage is that conversion between multiple currencies becomes a trivial matter. On the other hand, no "counter-based" privacy-protecting off-line cash system can prevent *undetectable* creation of new money in case a secured device is compromised. This clearly is a significant loss in security in comparison to the security offered by today's best off-line coin systems. The best that can be done is to limit the practical damages due to the compromise of a secured device.

In previously proposed counter-based off-line cash systems that protect privacy of payments, such as SmartCash [3] and Mondex [4], an attacker that manages to break the tamper-resistance of a secured device can greatly accelerate the rate by which new money can be introduced without detection into the system, by using accomplishes. Worse, anonymous publication of a description of the contents of a broken smart card can lead to a total break of the system. Without on-line verification, there is no practical way for the bank to limit the financial damage that can be done.

In Section 2, a counter-based off-line electronic cash system is proposed that offers appreciably greater security against the breaking of tamper-resistance than any other currently known counter-based off-line cash system. The new system is based on techniques that are applied in my off-line electronic coin system. The practicality of the system can be judged from the performance evaluation given in Section 3. Section 4 discusses in detail why the presented system is much more secure than any other counter-based privacy-protecting off-line cash system proposed thus far. Several practical measures for the bank are proposed as well, that should suffice in practice to guarantee that the cost of breaking a smart card will significantly exceed the expected financial profit that can be made from this. Conversion between multiple currencies is discussed in Section 5.

2 The system.

In the description of the system, the bank is denoted by $\mathcal{B}$ (multiple banks and clearing centers can easily be taken care of, and are hence not considered here), a generic user by $\mathcal{U}_i$, and a generic service provider by $\mathcal{S}_j$. $\mathcal{U}_i$ has at his disposal a computer, $\mathcal{C}_i$, that can be assumed to be under his own control: he can purchase it on the free market, or build it himself. It is assumed to have at least display and keyboard entry means, and may take the form of a palm top computer, a notebook, a Personal Computer, a TV remote control, and the like. $\mathcal{C}_i$ has a provision, such as a PCMCIA slot or a serial port, to interface with a smart card that will be issued to $\mathcal{U}_i$ when he opens an account. In addition, $\mathcal{C}_i$ may contain a PIN system, or perhaps even a biometric identification system, such that only $\mathcal{U}_i$, or ι party authorized by $\mathcal{U}_i$, can operate $\mathcal{C}_i$. It is assumed that the terminals of $\mathcal{B}$ and the service providers have suitable interface means for communicating with $\mathcal{C}_i$.

Appropriate software for performing the transaction protocols (described hereafter) is installed on $\mathcal{C}_i$ to ensure that payments are information-theoretically untraceable and unlinkable. This software need not be trusted by $\mathcal{B}$; $\mathcal{U}_i$ may retrieve it as public domain software, buy it on the free market, or write it himself. Preferably, it is part of user-friendly interface software that provides a suitable metaphor for conducting electronic cash transactions.

In the following description, all random numbers will be assumed to be "genuinely" random; Section 3 will address the use of pseudo-random generators to simulate these random numbers.

At several occasions, the terminology "inflow" and "outflow" in a protocol is used, referring to the presence in the protocol of a subliminal channel from $\mathcal{T}_i$ to the other party involved in the protocol (outflow) or vice versa (inflow). By interposing $\mathcal{C}_i$ between $\mathcal{T}_i$ and the other party involved in the protocol, such that $\mathcal{T}_i$ cannot directly communicate with the other party, $\mathcal{U}_i$ can prevent inflow and, more importantly, outflow.

The setup of the system. All arithmetic in the system is performed in a group G_q of prime order q for which polynomial-time algorithms are known to multiply, determine equality of elements, test membership, randomly select elements, and for which no feasible methods are known to compute discrete logarithms.

$\mathcal{B}$ generates independently at random two numbers $x, y \in \mathbb{Z}_q$, and a number $g_0 \in G_q \setminus \{1\}$. From now on, g_0^x will be denoted by h, and g_0^y by g_1. $\mathcal{B}$ also determines a collision-free hash function $\mathcal{H}(\cdot)$ that maps its inputs to $\mathbb{Z}_{2^k}$, where k is an appropriate security parameter. The function $\mathcal{H}$ is such that it is believed to make the Schnorr signature scheme [6] secure.

$\mathcal{B}$ also sets up an account database to store information about account holders, and a deposit database to store relevant information from deposited payment transcripts.

Opening an account. When $\mathcal{U}_i$ opens an account with $\mathcal{B}$, the following procedure takes place.

$\mathcal{B}$ generates independently at random a number $x_i \in \mathbb{Z}_q$. $\mathcal{B}$ lists $x + yx_i \bmod q$, henceforth denoted by I_i, in its account database together with other account information of $\mathcal{U}_i$ (comprising a balance that keeps track of the amount of money $\mathcal{U}_i$ has in its account with $\mathcal{B}$, and an accurate description of $\mathcal{U}_i$'s identity). I_i will be referred to as the identification number of $\mathcal{U}_i$. Suitable security measures must be taken by $\mathcal{B}$ to prevent attackers that gain access to the account database from modifying the balance of $\mathcal{U}_i$, or reading out the value of the identification number.

$\mathcal{B}$ issues to $\mathcal{U}_i$ a tamper-resistant smart card, $\mathcal{T}_i$, which has stored in non-volatile memory at least the following items:

- the number g_1 and the descriptions of G_q and $\mathcal{H}(\cdot)$;
- code to perform the smart card's role in the protocols;
- a counter, set to some appropriate initial value, that keeps track of the amount of money that is held by $\mathcal{U}_i$. This counter will be denoted by `balance`; and
- the number x_i.

$\mathcal{B}$ makes $g_1^{x_i}$, which will from now on be referred to as h_i, known to $\mathcal{U}_i$. $\mathcal{U}_i$ stores h_i in his computer $\mathcal{C}_i$, together with the numbers g_0, g_1, h and the descriptions of G_q and $\mathcal{H}(\cdot)$. $\mathcal{C}_i$ preferably also keeps track of a copy of `balance`, although this will not be explicitly mentioned in the description of the protocols.

The withdrawal protocol. Because $\mathcal{T}_i$ keeps track of the balance of $\mathcal{U}_i$ by means of a counter, $\mathcal{U}_i$ should only be allowed to pay an amount when the balance at payment time exceeds the amount to be paid; otherwise, $\mathcal{U}_i$ can pretend to have lost his smart card, once the balance has become negative. As a result, the balance in the smart card will on regular occasions have to be up-dated by means of a withdrawal protocol.

$\boxed{\texttt{Smart Card}}$ $\boxed{\texttt{Bank}}$

$\texttt{balance}' \leftarrow \texttt{balance}' - \texttt{amount}$
$v \leftarrow f(z, \texttt{seq}, \texttt{amount})$
$\texttt{seq} \leftarrow \texttt{seq} + 1$

$$\xleftarrow{v}$$

$v \overset{?}{=} f(z, \texttt{seq}, \texttt{amount})$
$\texttt{seq} \leftarrow \texttt{seq} + 1$
$\texttt{balance} \leftarrow \texttt{balance} + \texttt{amount}$

Figure 1: Withdrawal protocol.

Various designs for a suitable withdrawal protocol are conceivable. Note that no public-key cryptographic techniques are needed, since the contents of $\mathcal{T}_i$ are known to $\mathcal{B}$; the withdrawal protocol can hence be based on much more efficient conventional cryptographic techniques.

In the particular realization of a withdrawal protocol that is shown here, $\mathcal{T}_i$ is assumed to have in common with $\mathcal{B}$ a secret key z. This secret key, and a sequence number, $\texttt{seq}$, (which has been set to some initial value, such as zero), have been stored by $\mathcal{B}$ before issuing $\mathcal{T}_i$ to $\mathcal{U}_i$. In addition, the description of a one-way function $f(\cdot)$ has been stored by $\mathcal{B}$ in $\mathcal{T}_i$; this function may take the form of a block cipher, such as DES in encipherment mode, taking the secret key as the DES-key and the additional arguments as input. The function $f(\cdot)$ may even be kept secret by $\mathcal{B}$, for greater security. Of course, in practice $f(\cdot)$ may be deterministically related to $\mathcal{H}$, and z to x_i, for reasons of storage efficiency in $\mathcal{T}_i$.

To withdraw an amount, $\texttt{amount}$, $\mathcal{T}_i$ and $\mathcal{B}$ perform the following withdrawal protocol:

Step 1. $\mathcal{B}$ decreases the balance, $\texttt{balance}'$, of $\mathcal{U}_i$ by $\texttt{amount}$. It then increases $\texttt{seq}$ by one, and transfers $v \leftarrow f(z, \texttt{seq}, \texttt{amount})$ to $\mathcal{T}_i$.

Step 2. $\mathcal{T}_i$ computes $f(z, \texttt{seq}, \texttt{amount})$, and compares it for equality with v. If equality holds, it increases $\texttt{seq}$ by one, and $\texttt{balance}$ by $\texttt{amount}$.

Although another user cannot transfer money from the account of $\mathcal{U}_i$ with $\mathcal{B}$ to his own smart card, he can decrease the balance of $\mathcal{U}_i$ with $\mathcal{B}$ in this manner. Hence, $\mathcal{T}_i$ should first identify itself to $\mathcal{B}$ before the protocol is executed. Hereto, a similar protocol, with the roles of $\mathcal{T}_i$ and $\mathcal{B}$ interchanged, can be used. Note that such a protocol can also be used to enable $\mathcal{U}_i$ to deposit money from his smart card to his account.

Remarks. 1. Alternatively, $\mathcal{U}_i$ may interpose his computer $\mathcal{C}_i$, to prevent inflow and/or outflow. To ensure that the number transferred by $\mathcal{B}$ in Step 1 does not contain inflow, $\mathcal{C}_i$ may require $\mathcal{T}_i$, before passing v on to it, to provide a commit on $f(z, \texttt{seq}, \texttt{amount})$. After $\mathcal{C}_i$ has passed v on to $\mathcal{T}_i$, $\mathcal{T}_i$ opens the commit to reveal that has been able to compute v all along, thereby proving that there has been no inflow. The commitment function can be based on a block-cipher, taking for instance $f(z, \texttt{seq}, \texttt{amount})$ and a random number as inputs.

2. The increase of the sequence number, $\texttt{seq}$, serves to prevent a replay attack by $\mathcal{U}_i$. The increment by one has been chosen for explicitness; alternatively, $\mathcal{B}$ may specify the new sequence number to be determined from the old sequence number according to a more complicated relation, which may be kept secret.

3. Instead of letting $\mathcal{B}$ keep track of the value of $\texttt{seq}$ for each smart card, $\mathcal{C}_i$ or $\mathcal{T}_i$ can inform $\mathcal{B}$ at withdrawal time of the current value of $\texttt{seq}$.

The certificate issuing protocol. Payment of an amount requires $\mathcal{U}_i$ to provide the payee with a signature on the amount (and additional data to prevent a replay attack). This signature is made with respect to a public key that must have been certified by $\mathcal{B}$. A certificate of $\mathcal{B}$ on a public key may only be used once in a payment. Double-spending a certificate requires $\mathcal{U}_i$ to break the tamper-resistance of $\mathcal{T}_i$, and enables $\mathcal{B}$ to afterwards identify $\mathcal{U}_i$. It is this mechanism, inherited from privacy-protecting off-line coin systems, that gives the presented system a much higher security level than other proposals for counter-based systems.

To prepare for the withdrawal of a certificate, $\mathcal{T}_i$ and $\mathcal{C}_i$ perform the following off-line pre-processing:

Step 1. $\mathcal{T}_i$ generates independently at random a number $w_i \in_{\mathcal{R}} \mathbb{Z}_q$, and sends $a_i \leftarrow g_1^{w_i}$ to $\mathcal{C}_i$. $\mathcal{T}_i$ stores w_i for later use in the payment protocol.

Step 2. $\mathcal{C}_i$ generates independently at random a vector $(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5) \in (\mathbb{Z}_q)^5$. It then computes $h'_i \leftarrow h_i g_0^{\alpha_1}$, $a'_i \leftarrow a_i g_1^{\alpha_2} g_0^{\alpha_3}$, and $\texttt{temp} \leftarrow g_0^{\alpha_4}(hh_i)^{\alpha_5}$. $\mathcal{C}_i$ stores h'_i, a'_i and $(\alpha_1, \alpha_2, \alpha_3)$ for later use in the payment protocol, and temporarily stores $\texttt{temp}$, α_4, and α_5.

In practice, $\mathcal{T}_i$ and $\mathcal{C}_i$ will prepare for withdrawal of a batch of certificates, by pipe-lining many executions of this pre-processing phase.

The actual withdrawal of the certificate is done by means of the following on-line certificate issuing protocol between $\mathcal{C}_i$ and $\mathcal{B}$:

Step 1. $\mathcal{B}$ generates at random a number $w \in \mathbb{Z}_q$, and sends $a = g_0^w$ to $\mathcal{C}_i$.

Step 2. $\mathcal{C}_i$ computes $c' = \mathcal{H}(h'_i, a'_i, a\,\texttt{temp})$. It stores c', and sends $c = c' + \alpha_5 \bmod q$ to $\mathcal{B}$.

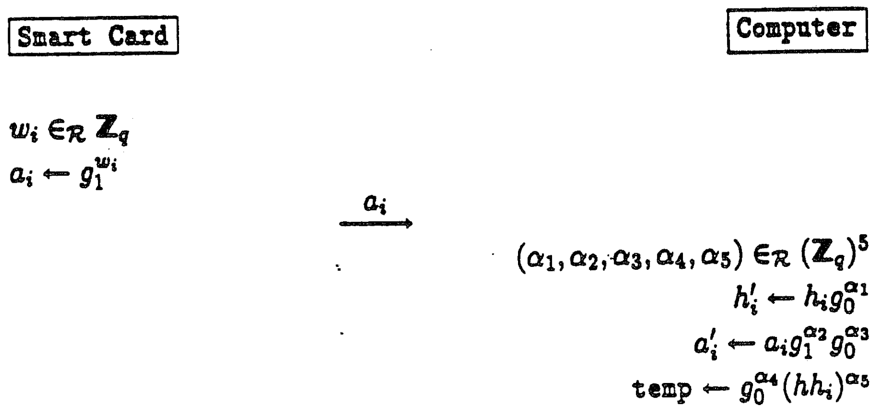
Smart Card
Computer
$w_i \in_{\mathcal{R}} \mathbb{Z}_q$
$a_i \leftarrow g_1^{w_i}$
$\xrightarrow{a_i}$
$(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5) \in_{\mathcal{R}} (\mathbb{Z}_q)^5$
$h_i' \leftarrow h_i g_0^{\alpha_1}$
$a_i' \leftarrow a_i g_1^{\alpha_2} g_0^{\alpha_3}$
$\texttt{temp} \leftarrow g_0^{\alpha_4}(hh_i)^{\alpha_5}$

Figure 2: Pre-processing for the certificate issuing protocol.

Step 3. $\mathcal{B}$ sends $r = cI_i + w \bmod q$ to C_i.

Note that no participation whatsoever of T_i is needed in the on-line part of the certificate issuing protocol. As will be appreciated, the computational requirement for C_i is very small; it is virtually equal to the effort required for one modular multiplication of two 64 byte numbers.

C_i can now go off-line again, and perform the following post-computation:

> C_i verifies that $g_0^r(hh_i)^{-c} = a$. If the verification holds, it computes $r' = r + c'\alpha_1 + \alpha_4 \bmod q$, and stores r'. The numbers `temp`, α_4, and α_5 can be erased; they are no longer needed.

In practice, an occasional verification of r at randomly chosen executions of the post-processing phase will be sufficient, since if the verification does not hold then the payee will not accept in the payment protocol. Furthermore, it can be proven that $\mathcal{B}$ cannot send an r such that T_i in the corresponding payment protocol can leak more than one bit of outflow. Therefore, a scenario in which the payees cooperate with $\mathcal{B}$ to gather tracing information, by also accepting incorrectly formed certificates, will definitely not be beneficial to $\mathcal{B}$ (it enables users to create their own certificates, which can be spent without a corresponding decrease of the counter in their smart cards).

The payment protocol. To pay to S_j an amount, `amount`, T_i and C_i perform the following pre-processing:

Step 1. C_i sends $(h_i', a_i', \texttt{specification})$ to T_i. Here, `specification` is a concatenation, in a standardized format, of an `amount` field, a `time` of transaction field, a `date` of

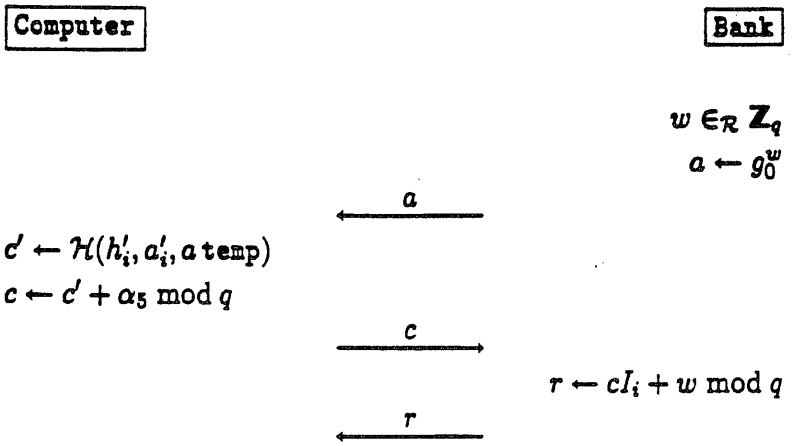
Computer
Bank
$w \in_{\mathcal{R}} \mathbf{Z}_q$
$a \leftarrow g_0^w$
a
$c' \leftarrow \mathcal{H}(h'_i, a'_i, a\,\texttt{temp})$
$c \leftarrow c' + \alpha_5 \bmod q$
c
$r \leftarrow cI_i + w \bmod q$
r

Figure 3: On-line part of the certificate issuing protocol.

transaction field, and a field for the account number of $\mathcal{S}_j$ with $\mathcal{B}$ (or some other information in a format determined by $\mathcal{B}$ and uniquely associated with $\mathcal{S}_j$). Additional information fields may be included in **specification**.

Step 2. $\mathcal{T}_i$ verifies that w_i is still in memory, and that **balance** exceeds **amount**. If this is the case, it computes $d \leftarrow \mathcal{H}(h'_i, a'_i, \texttt{specification})$, and $r_{i1} \leftarrow dx_i + w_i \bmod q$. It then decreases **balance** by **amount**, erases w_i from memory, and sends r_{i1} to $\mathcal{C}_i$.

Step 3. $\mathcal{C}_i$ also computes $d \leftarrow \mathcal{H}(h'_i, a'_i, \texttt{specification})$, and verifies that $g_1^{r_{i1}}(h'_i)^{-d} = a_i$ (as in the certificate issuing protocol, this verification may be done only occasionally, or at a later stage). If this is the case, then $\mathcal{C}_i$ computes $r'_{i1} \leftarrow r_{i1} + \alpha_2 \bmod q$ and $r_{i2} \leftarrow d\alpha_1 + \alpha_3 \bmod q$.

Note that it has been assumed that $\mathcal{U}_i$ (or $\mathcal{C}_i$) can determine **specification** without assistance of $\mathcal{S}_j$. If the system is used for an application such as payments over a computer network, this is very plausible; date and time can be looked up (they may even be entered manually by $\mathcal{U}_i$ by using the keyboard of $\mathcal{C}_i$, although typically $\mathcal{C}_i$ will retrieve this information locally), and the account number of $\mathcal{S}_j$ may be stored on a local server, or stored on CD-ROM.

The actual payment is done by means of the following on-line payment protocol between $\mathcal{C}_i$ and $\mathcal{S}_j$:

$\mathcal{C}_i$ sends $h'_i, (c', r'), (a'_i, r'_{i1}, r_{i2})$ to $\mathcal{S}_j$.

$\mathcal{S}_j$ computes $d \leftarrow \mathcal{H}(h'_i, a'_i, \texttt{specification})$, and accepts the transferred information if and only if $c' = \mathcal{H}(h'_i, a'_i, g_0^{r'}(hh'_i)^{-c'})$ and $g_1^{r'_{i1}} g_0^{r_{i2}}(h'_i)^{-d} = a'_i$.

$\boxed{\text{Smart Card}}$ $\boxed{\texttt{Computer}}$

$$\xleftarrow{(h'_i, a'_i, \texttt{specification})}$$

$\texttt{balance} \overset{?}{\geq} \texttt{amount}$

w_i present?

$d \leftarrow \mathcal{H}(h'_i, a'_i, \texttt{specification})$

$r_{i1} \leftarrow dx_i + w_i \bmod q$

$\texttt{balance} \leftarrow \texttt{balance} - \texttt{amount}$

Erase w_i

$$\xrightarrow{r_{i1}}$$

$d \leftarrow \mathcal{H}(h'_i, a'_i, \texttt{specification})$

$g_1^{r_{i1}}(h'_i)^{-d} \overset{?}{=} a_i$

$r'_{i1} \leftarrow r_{i1} + \alpha_2 \bmod q$

$r_{i2} \leftarrow d\alpha_1 + \alpha_3 \bmod q$

Figure 4: Pre-processing for the payment protocol.

It is conceivable that $\mathcal{S}_j$ expects another `time` field value to have been used by $\mathcal{U}_i$, since it can be expected that the granularity of the `time` field in `specification` is such that small disturbances in clock synchronization between $\mathcal{U}_i$ and $\mathcal{S}_j$ result in different values. As will be appreciated, this is not a problem: in the on-line part of the payment protocol, $\mathcal{U}_i$ can send the chosen value for the `time` field along. Given that in the deposit protocol $\mathcal{B}$ will accept the payment transcript as long as it is not identical to one that $\mathcal{S}_j$ deposited before, $\mathcal{S}_j$ need merely check that $\mathcal{U}_i$ does not re-use the same certificate later on for the same suggested value for the `time` field. To this end, $\mathcal{S}_j$ can for example accept any suggested value for the `time` field that lies within a certain time span, and at each payment check the involved payment data for equality against the transcripts that have been received in this time span. A time span of, say, 15 minutes should be more than sufficient for most practical applications.

In Section 3, optimizations are discussed that reduce the amount of data that must be transferred from $\mathcal{C}_i$ to $\mathcal{S}_j$.

If it desired that $\mathcal{T}_i$ communicate directly with $\mathcal{S}_j$, for instance because $\mathcal{S}_j$ has no suitable interface means for communicating with $\mathcal{C}_i$, then $\mathcal{U}_i$ can provide the information computed in Step 3 to $\mathcal{T}_i$. $\mathcal{T}_i$ can then pass it on to $\mathcal{S}_j$. In practice, this amounts to $\mathcal{U}_i$ first having to insert his smart card into his computer, and then into the interface with $\mathcal{S}_j$. (Giving $\mathcal{T}_i$ the

$$\boxed{\texttt{Computer}} \qquad\qquad\qquad\qquad \boxed{\texttt{Service Provider}}$$

$$\xrightarrow{h'_i, (c', r'), (a'_i, r'_{i1}, r_{i2})}$$

$$\begin{aligned} d &\leftarrow \mathcal{H}(h'_i, a'_i, \texttt{specification}) \\ c' &\overset{?}{=} \mathcal{H}(h'_i, a'_i, g_0^{r'}(hh'_i)^{-c'}) \\ g_1^{r'_{i1}} g_0^{r_{i2}} (h'_i)^{-d} &\overset{?}{=} a'_i \end{aligned}$$

Figure 5: On-line part of the payment protocol.

abilities to perform the role of $\mathcal{C}_i$ as well, puts to much computational burden on $\mathcal{T}_i$, and $\mathcal{U}_i$ cannot verify that his privacy is protected.)

Alternatively, if $\mathcal{B}$ has also stored a certificate on h_i in $\mathcal{T}_i$, then $\mathcal{T}_i$ can transfer cash to $\mathcal{S}_j$ without any assistance of $\mathcal{C}_i$: it hereto transfers h_i, the certificate on h_i, and a Schnorr signature on `specification`, made with respect to h_i, to $\mathcal{S}_j$. Of course, all payments made in this way are linkable and traceable to $\mathcal{U}_i$.

The deposit protocol. At a suitable time, preferably when network traffic is low, $\mathcal{S}_j$ sends the payment transcript, consisting of $h'_i, (c', r'), (a'_i, r'_{i1}, r_{i2})$ and `specification`, to $\mathcal{B}$.

$\mathcal{B}$ verifies that `specification` has been formed correctly by $\mathcal{S}_j$, and computes $d \leftarrow \mathcal{H}(h'_i, a'_i, \texttt{specification})$. $\mathcal{B}$ then searches its deposit database to find out if $\texttt{hash}_1(h'_i, c', r')$ has already been stored. Here, $\texttt{hash}_1(\cdot)$ is a simple hash function, that need not be one-way (it's use is merely to speed up the search process). There are two possible situations:

1. $\texttt{hash}_1(h'_i, c', r')$ is not yet in the deposit database. $\mathcal{B}$ then verifies the payment transcript by verifying that $c' = \mathcal{H}(h'_i, a'_i, g_0^{r'}(hh'_i)^{-c'})$ and $g_1^{r'_{i1}} g_0^{r_{i2}} (h'_i)^{-d} = a'_i$. Only if the verification holds, $\mathcal{B}$ stores $(\texttt{hash}_1(h'_i, c', r'), \texttt{specification}, d, r'_{i1})$ in its deposit database as having been deposited by $\mathcal{S}_j$, and credits the account of $\mathcal{S}_j$ by `amount`.

2. $\texttt{hash}_1(h'_i, c', r')$ is already in the deposit database. In that case with overwhelming probability a fraud has occurred. If the `specification` of the already stored information is identical to that of the new payment transcript, then $\mathcal{S}_j$ is trying to deposit the same transcript twice.

   Otherwise, $\mathcal{B}$ verifies the transcript as described in situation 1. If the verification holds (the payment transcript is valid), the certificate (c', r') must have been double-spent with overwhelming probability (if not, a very coincidental collision under $\texttt{hash}_1(\cdot)$ has occured, which will be detected with overwhelming probability by $\mathcal{B}$ after the following computation). Since $\mathcal{B}$ now has at its disposal a pair (d, r'_{i1}) from the new transcript and a pair (d', r''_{i1}) from the already deposited information, it can compute $x+y(r'_{i1}-r''_{i1})/(d-d') \bmod q$. $\mathcal{B}$ then searches its account database for this identification number; in case the certificate was double-spent, the corresponding account holder can be identified, and appropriate legal actions can be taken.

3 Performance evaluation

For explicitness, it will be assumed that G_q is the subgroup of $\mathbf{Z}_p^*$ for some prime p such that $q|(p-1)$ (although it is good to realize that an elliptic curve implementation can significantly reduce the storage required for the public keys of $\mathcal{B}$). Using the parameter lengths proposed by Schnorr [6] for his signature scheme, it will be assumed that $k = 72$, $|q| = 140$, and $|p| = 512$, where $|\cdot|$ denotes binary length. For greater security, one may want to increase these values.

Three practical optimizations will be made. Firstly, in line with an earlier remark, it will be assumed that $\mathcal{C}_i$ in the certificate issuing protocol leaves out the post-processing verification $g_0^r(hh_i)^{-c} = a$.

Secondly, it will be assumed that both $\mathcal{C}_i$ and $\mathcal{T}_i$ simulate the generation of random numbers by pseudo-random numbers that are generated by iteratively applying some suitable deterministic function. Since all the required random numbers are only 140 bits in size, it is reasonable to assume that the generation of each such number requires a computational effort significantly less than a multiplication modulo a 512 bit prime. For example a linear congruential generator, or the exclusive-or of several thereof, might be used by $\mathcal{C}_i$, and a DES-based one-way function by $\mathcal{T}_i$. (Note that the randomness of $\mathcal{C}_i$ only serves to achieve privacy of payments, whereas that of $\mathcal{T}_i$ serves security of the system; by trading off with efficiency, the bank may prefer to let $\mathcal{T}_i$ use a cryptographically strong pseudo-random number generator.)

Thirdly, it will be assumed that d in the payment and deposit protocol is computed by all parties involved as $d \leftarrow \mathcal{H}(h'_i, \mathtt{hash}_2(a'_i), \mathtt{specification})$. Here, $\mathtt{hash}_2(\cdot)$ is a hash-function; it need not be one-way, although this certainly cannot harm. For concreteness, it will be assumed that its arguments are mapped to 140 bit numbers (although 72 bits might suffice). The verification relation $g_1^{r'_{i1}} g_0^{r_{i2}} (h'_i)^{-d} = a'_i$ must correspondingly be replaced by $\mathtt{hash}_2(g_1^{r'_{i1}} g_0^{r_{i2}} (h'_i)^{-d}) = \mathtt{hash}_2(a'_i)$. A further optimization to reduce the size of the transmitted data in the payment and deposit protocol will be discussed below; it is not taken into account in the following performance estimate, though.

As far as computational speedups is concerned, the performance estimate is fairly unsophisticated: only the computation of products such as $g_1^a g_0^b$ using straightforward simultaneous repeated squaring is considered, since the goal of this section is merely to compare the performance of the system with the requirements for a full RSA encryption with the same length of the modulus[1]. The computational effort of generating the random numbers will be neglected, in view of the second optimization. Likewise, the computational effort of computing images of $\mathcal{H}(\cdot)$ and $f(\cdot)$ is neglected, given that these functions can be implemented much more efficiently than a modular multiplication of two 64 byte numbers.

The following list shows the estimated performance figures for each of the four types of participants in the basic system:

* $\mathcal{C}_i$ must permanently store (g_0, g_1, h), (p, q), h_i, the descriptions of $\mathcal{H}(\cdot)$ and the pseudo-random number generator, and an initial seed for the pseudo-random number generator. $\mathcal{C}_i$ also needs a software program to perform its role in the protocols.

  In order to withdraw a certificate, the most demanding part of the system, $\mathcal{C}_i$ must perform about 700 multiplications modulo a 64 byte number, which is less than the effort required to perform one full RSA exponentiation. Virtually all computations can be done off-line in a precomputing phase; the on-line effort is comparable to only one modular multiplication of two 64 byte numbers.

  Assuming that $\mathcal{C}_i$ regenerates $\alpha_1, \alpha_2, \alpha_3$, and h'_i, at payment time, the dynamic storage for a certificate, consisting of (c', r') and $\mathtt{hash}_2(a'_i)$, is merely 44 bytes, plus about one byte to keep track of the state of the pseudo-random number generator (this should be sufficient: it allows regeneration of the number h'_i for about 85 certificates without needing to update the seed).

  In the payment protocol, $\mathcal{C}_i$ needs to send, amongst others, the number h'_i. Regenerating this number requires about 210 multiplications modulo p, which is less than one third of the cost for a full RSA exponentiation. At the cost of storing also h'_i at withdrawal time, the computational requirements for $\mathcal{U}_i$ in the payment protocol become virtually zero. In any case, all computations can be pre-processed: no real-time computation is required.

  In total, $\mathcal{U}_i$ needs to send only 143 bytes in order to pay. Below, it is shown how the size of data transfer can be compressed further to a mere 125.5 bytes.

* $\mathcal{T}_i$ must permanently store g_1, (p, q), x_i, the descriptions of a pseudo-random number generator, and an initial seed for the pseudo-random number generator.

  The requirements for $\mathcal{T}_i$ to assist in withdrawing and spending of a certificate are exactly those of the signer in the Schnorr identification scheme. Specifically, to assist in

[1] A full RSA encryption is the computation of $X^e \bmod n$ for a random X and unknown prime factorization of n, such that the binary length of e is about equal to that of n. In an RSA-based encryption scheme, this is the effort required to encrypt in case the decryption exponent is chosen small so as to be able to decrypt very fast.

withdrawing a certificate $\mathcal{T}_i$ must generate one 17.5 byte number, and perform about 210 multiplications modulo p. To assist in spending the certificate, it must regenerate this 17.5 byte number from its pseudo-random number generator and compute one response at virtually no computational cost. All these computations can be pre-processed off-line.

* The computational effort of $\mathcal{S}_j$ in the payment protocol is about the same as that of $\mathcal{B}$ in the deposit protocol: slightly over 500 modular multiplications of two 64 byte numbers. This is equal to about two thirds of the cost of one full RSA exponentiation.

* The computational effort of $\mathcal{B}$ in the certificate issuing protocol is the same as that of the signer in the Schnorr signature protocol: less than two thirds of the cost of one full RSA exponentiation. Storage of $(\texttt{hash}_1(h'_i, c', r'), \texttt{specification}, d, r'_{i1})$ for a deposited payment transcript requires only 50 bytes, if it is assumed for convenience that the hash-function, $\texttt{hash}_1(\cdot)$, maps its input to a 48-bit number and `specification` is 17.5 bytes long. Applying the verification relation for a certificate requires just over 500 multiplications modulo p, which is about two thirds of the cost of one full RSA exponentiation.

As mentioned, a further optimization is possible to reduce the size of the transferred data in the payment and deposit protocol. Although I have not been able to reduce the resulting security to a natural assumption, it is my conviction that this further optimization can be applied without loss in security. The optimization is as follows: in order to prevent the sending of a'_i or $\texttt{hash}_2(a'_i)$ completely, d is determined as $d \leftarrow \mathcal{H}(h'_i, (c', r'), \texttt{specification})$ (caution must be taken here: leaving out (c', r') from the hash-value has the effect that valid payment transcripts, with arbitrary `specification`, can be forged). $\mathcal{S}_j$ in the payment protocol, and $\mathcal{B}$ in the deposit protocol, must correspondingly verify the correctness of the payment data by comparing c' for equality to $\mathcal{H}(h'_i, g_1^{r'_{i1}} g_0^{r_{i2}} (h'_i)^{-d}, g_0^{r'} (hh'_i)^{-c'})$. As will be appreciated, this optimization also has the effect of significantly reducing the dynamic storage requirements for a certificate: since a'_i need no longer be stored by $\mathcal{C}_i$ a certificate can be stored in merely 26.5 bytes. Moreover, only 125.5 bytes must be transferred in the payment protocol from $\mathcal{C}_i$ to $\mathcal{S}_j$ to pay any specified amount.

4 Practical considerations

In previously proposed counter-based privacy-protecting off-line cash systems, an attacker that manages to break the tamper-resistance of a smart card can introduce new money into the system without ever needing to visit the bank. The main cause for concern is that such an attacker can greatly accelerate the rate of financial profit by cooperating with many accomplishes. The attacker can even screw up the entire system by, say, anonymously publishing the contents of the broken smart card, since there is no way for the bank to ever identify him. Without on-line verification, there is no practical way for the bank to limit the damage that can be done. This clearly is a dramatic loss in security in comparison to the security offered by the currently best coin systems.

In the system proposed in this paper, a successful attacker that manages to break a smart card cannot spend the same certificates over and over again without being identified afterwards by the bank. The only way for a successful attacker of a smart card to introduce new money into the presented system, without becoming traceable by the bank, is by withdrawing new certificates. Since the bank can take appropriate actions that prevent withdrawal of an "unlimited" number of certificates, the expected damage due to a compromised smart card will be greatly reduced. It is this feature, that in effect excludes the use of accomplishes, that makes the presented system much more secure than all other known counter-based privacy-protecting off-line cash systems.

To further limit the expected financial damage due to a broken smart card, the bank can take some or all of the following pre-cautionary measures:

- Certificates can be assigned a specific maximum spendable value to; payment amounts that are in excess of this maximum amount must be paid using more than one certificate.

- In combination with the previous measure, the bank can charge the withdrawer for the maximum spendable value of a withdrawn certificate. For the unspent part, a refund can be requested by the smart card. (A protocol similar to the withdrawal protocol can be used for this—note that the withdrawal protocol between T_i and B becomes obsolete).

- In combination with the preceding measure, the bank can limit the amount for which refunds may be requested, and limit the time period in which a refund for withdrawn certificates can be requested. For instance, a certificate will only be refunded for up to 30 percent of its maximum value.

- The bank can use certificates in combination with coins. Certificates are assigned a fairly low maximum value to, and so will in practice be used to pay with only if too many coins would be needed otherwise. For instance, \$16,39 might be paid using one \$10 coin, one \$5 coin, and one certificate that has a maximum value of \$2 and is spent for \$1,39.

  Given that in my coin system the withdrawal of a certificate is technically exactly the same as the withdrawal of a certificate, this combination is particularly appealing.

- The bank can limit the number of certificates that can be withdrawn per account in a given period of time.
- The bank can physically scrutinize the smart card on a regular basis, perhaps specifically concentrating on users that withdraw suspiciously many certificates.

  Note that automated physical verification at withdrawal time is very cumbersome in practice, and can only be superficial. An attacker that can break and extract the contents of a smart card can also be expected to be able to make an adequate copy that will pass such physical verification.

5 Conversion between multiple currencies.

As noted, an advantage of counter-based systems over coin systems is that conversion between multiple currencies is a trivial matter. To transfer an amount that is specified in a different currency than the local currency of a smart card, the amount is first multiplied by the smart card by an appropriate conversion rate to convert it into the local currency, before subtracting it from, or adding it to, its counter. There are several ways for the bank to ensure use of valid conversion rates:

- A field is reserved in `specification` that indicates the two currencies involved in the payment, and the conversion rate used for these two currencies. This field can be filled in at payment time by the parties involved in the payment (in a practical situation, this means that $\mathcal{U}_i$ will have to inform C_i of whether to accept the conversion rate or not). Since `specification` also consists of `time` and `date` of the transaction, the bank can verify at deposit time whether a correct conversion rate has been applied to the two currencies specified in `specification`. If not, it rejects the payment transcript. (Of course, the bank only has to see to it that the conversion rate that has been applied is not to its disadvantage; if $\mathcal{U}_i$ at payment time is willing to accept a currency conversion rate that is unfavourable to him, that's his own business.)

  This leaves open the possibility that $\mathcal{U}_i$ and S_j together decide at payment time decide on `time` and `date` values which are favourable to them. This is not a problem if deposits have to be made, say, at the end of the day. Alternatively, the bank ensures that the terminal of S_j is secured, so it always generates the proper date.

- The bank can encode a list of currency rates into the certificates it issues. The number h_i' is taken of the form $g_0^{\alpha_1} g_1^{I_i} g_2^{l}$, where l is an encoding of the conversion rates list, and g_2 is another publicly known generator of G_q that has been chosen by the bank. The bank can encode l into h_i' by using the technique used for encoding I_i into h_i'. At payment time, C_i reveals l to S_j, from which the conversion rate between the twocurrencies involved is selected. Both parties then set h'_i to h'_i/g_2^l; the rest of the payment protocol remains the same as before. As above, `specification` requires a field that indicates the two currencies involved in the payment, and the conversion rate used for these two currencies.

6 Concluding remarks.

Assuming that the tamper-resistance of the smart cards cannot be broken, the system is provably as hard to break as the Schnorr signature scheme. The build-in mechanism for traceability of a double-spent certificate is as hard to break as the blinded Schnorr signature scheme [5]. The user-controlled computer ensures that payments are information-theoretically untraceable and unlinkable. The proofs of are omitted in this paper, since they are almost the same as the proofs for my coin system.

The privacy offered by my coin system is better than that offered by the counter-based system presented here: in the coin system, payments of a user cannot be traced even if the bank can examine the contents of all smart cards that have been used by its account holders to conduct transactions with (the bank cannot even learn the amounts that have been paid). Since the security of my coin system also is significantly greater than that of the counter-based system, and storage and computation capabilities of small computing devices rapidly grow as technology progresses, coin systems may ultimately provide the most satisfactory solution for off-line electronic cash systems.

Since the counter-based system is from a technical point of view almost the same as my coin system, a well-known generally applicable method for transferability in off-line coin systems, described by Van Antwerpen [1], can also be applied in the counter-based system. The practicality of this method for smart card implementation is not clear, though: the computational requirements for verifying a certificate that is transferred for the k-th time are roughly equal to the effort of computing $2k/3$ full RSA exponentiations. Hence, in practice a fairly small bound will have to be imposed on the number of times that cash can be transferred from smart card to smart card. Whether building in this feature is really worthwhile will depend on the application in which the payment system is put to use.

The axis of the construction presented in this paper is a new technique for constructing certificate issuing and showing protocols, which is inherited from my coin system. As will be appreciated, this technique can be applied to any known Fiat/Shamir type signature scheme. The explicit choice for the Schnorr signature scheme has merely been made because it provides the most efficient realization.

References

[1] Antwerpen, H., "Electronic cash," Master's thesis Univ. Eindhoven, 1990.

[2] Brands, S., manuscript (1993), submitted in parts for publication. A preliminary version can be found in "Untraceable Off-Line Cash in Wallets With Observers," Crypto '93, LNCS 773, Springer-Verlag, pages 302–318.

[3] Bos, J. and Chaum, D., "SmartCash: a practical electronic payment system," Technical Report CS-R9035, CWI, Amsterdam, August 1990.

[4] "UK banks introduce Mondex, the cashless cash card," Newsbytes News Network, January 6, 1993.

[5] Okamoto, T., "Provably secure and practical identification schemes and corresponding signature schemes," Crypto '92, Springer-Verlag, pages 31–53.

[6] Schnorr, C.P., "Efficient Signature Generation by Smart Cards", Journal of Cryptology, Vol. 4, No. 3, (1991), pages 161–174.