An Efficient Fair Off-Line Electronic Cash System with Extensions to Checks and Wallets with Observers

Aymeric de Solages and Jacques Traoré

France Télécom - Branche Dévéloppement
Centre National d'Etudes des Télécommunications
42, rue des Coutures, BP 6243
14066 Caen Cedex, France
{aymeric.desolages, jacques.traore}@cnet.francetelecom.fr

Abstract. In this paper, we present a privacy-protecting off-line electronic cash system which is *fair*, that is, the transactions are (potentially) traceable by a trusted authority but anonymous otherwise. Our scheme, based on a modification of Brands'restrictive blind signature scheme [2], is significantly more efficient than that of [11], while offering the same functionalities (*off-line* trusted authority, *direct* identification of the owner[1] of a coin when the tracing of a user from his coin is performed by the trusted authority). Furthermore, we show how to extend our system to wallets with *observers* [9] and to electronic *checks* [1, 2, 15]. These two extensions are more efficient than previous ones [2, 6]. The first extension is featured by a high computational efficiency and low storage requirements for observers. The second extension provides checks which are more efficiently computed than checks in [2] (twice as fast) and which also require less memory for their storage (half as much).

1 Introduction

Current cashless payment systems such as credit card payment systems provide little or no protection of the users' privacy. Indeed, in these systems, the banks could easily observe who pays which amount to whom and when.

At Crypto'82, D. Chaum [7] introduced a new cryptographic tool, the blind signature scheme, which has made it possible to design anonymous (privacy-protecting) prepaid payment systems [2, 3, 8]. A blind signature scheme is a cryptographic protocol involving two entities: a sender and a signer. This protocol allows the sender to choose a message and obtain a digital signature of this message from the signer without revealing anything about the contents of the message to the signer. Moreover, the signer cannot link later on (i.e. after the signature has been

[1] By owner of a coin, we mean in fact the person who has withdrawn the coin.

revealed to the public) a given message-signature pair to the corresponding execution of the blind signature protocol.

Recent anonymous prepaid electronic payment systems [2, 3, 8], based on the blind signature technique, 'emulate' physical cash. In these systems, the users withdraw electronic coins which consist of numbers, generated by users, and *blindly* signed by an electronic money issuer (a bank). Each signature represents a given amount. These coins are then spent (released) in shops which can authenticate them by using the public signature key of the bank. In these systems it is impossible to link a withdrawal of an electronic coin to a payment made with this coin (perfect anonymity).

Unfortunately, these perfect anonymous payment systems could be abused for unlawful and criminal activities, such as money laundering and *perfect* blackmailing [21]. For these reasons, it has been argued that perfect anonymity is certainly not the suitable level of privacy that an electronic payment system must offer to its users. Future electronic cash systems should be provided with incomplete or conditional anonymity.

A step in this direction has recently been made by several researchers. In [19], Stadler et al. proposed a new type of blind signature scheme called *fair Blind Signature Scheme* (*fair BSS* in short), that can replace ordinary BSS within anonymous payment systems. In a fair BSS, the signer can, with the help of a single (or several) trusted authority (ies) (who is (are) given some extra information), either link a message-signature pair to the corresponding execution of the signing protocol or extract the content of a message from its *blind* form. By replacing *ordinary* blind signature schemes by *fair* blind signature schemes in electronic payment systems, the bank and anyone else would still be unable to link a withdrawal to the payment made with the withdrawn coin. However, if abuse is suspected, the trusted authority could help the bank in auditing a particular account or to finding out the author's identity in one particular transaction. In this way, privacy of honest users would be preserved and embezzlement by criminals prevented. Unfortunately the schemes of Stadler et al. are inefficient. Moreover, one of their schemes has been recently broken by one of the authors of this paper [20].

In [4], Brickell, Gemmell and Kravitz proposed an anonymous off-line electronic cash scheme, based on Brands' system [3], which is fair, insofar as the transactions are (potentially) traceable by proper trusted authorities but anonymous otherwise. In this scheme, the trusted authorities are *on-line*, that is, they are involved[1] in every withdrawal so that they are able to revoke (if requested) the user's anonymity.

Recently, Camenisch, Maurer and Stadler proposed a more efficient fair off-line electronic cash system [5]. Moreover, in their scheme, the trusted authorities are *off-line*, that is, they need not be involved in either the withdrawal protocol or the payment protocol in order to be able to revoke the user's anonymity. In [6], Camenish et al. showed how to extend their basic fair cash system [5] to wallets with observers.

[1] In fact, the trusted authority can pre-compute his involvement in the withdrawal.

In [14], Frankel, Tsiounis and Yung achieved similar results to [5], but with a stronger model of fair cash: in [14], the trusted authority finds directly the identity of the owner of a specific coin, whereas in [5] the trusted authority performs a search in a large database (the withdrawal database).

In [11], Davida et al. improved the efficiency of [14].

In this paper, we propose an efficient fair off-line electronic cash system. Our scheme, based on a modification of Brands'restrictive blind signature scheme [3], is significantly more efficient than that of [11], while offering the same functionalities (*off-line* trusted authority, *direct* identification of the owner of a coin when the tracing of a user from his coin is performed by the trusted authority).

Moreover, we show how to extend our system to wallets with *observers* [9] and to electronic *checks* [1, 2, 15]. These two extensions are significantly more efficient than previous ones [1, 6].

Organization of the paper: In section 2, we explain some notations and introduce our assumptions and the background of the key techniques that will be useful in the sequel. In section 3, our generic fair cash system is described, followed by a discussion on its security. We also compare the efficiency of our system with the system of [11]. In section 4, we show how to extend our generic fair cash system to the setting of wallets with observers. Then, we examine the security of this extension and compare its efficiency with [6]. In section 5, we show how our system can be extended to handle electronic checks. In section 6, we conclude this paper and introduce some open problems.

In the following sections, we describe a generic fair off-line payment system. In the simplified model of off-line electronic cash system that we use, three types of parties are involved: the customers (or 'users'), the shops and a bank. Three possible transactions may occur between them: the withdrawal (by a user from the bank), the payment (by a user to a shop), and the deposit (by a shop to the bank). In the withdrawal protocol, the user withdraws electronic coins from the bank while his account is being debited. In the payment protocol, the user pays the shop with the coins he has withdrawn. In the deposit protocol, the shop deposits the coins it has received to the bank and the shop's account is credited.

Our scheme is an anonymous payment system, however the customers' anonymity may be revoked by a proper trusted authority. The customers' anonymity can be revoked in two different ways:

- *coin tracing:* the bank provides the trusted authority with the data of withdrawals of a (suspect) user and asks for the information that allows it to determine the corresponding deposits (or payments).
- *owner tracing:* the bank provides the trusted authority with data of a (suspect) payment (in fact the deposit) and asks for the identity of the customer who has withdrawn the money used in this (suspect) payment.

2 Notations, Assumptions and Basic Tools

The security of our schemes is based on assumptions about the difficulty of solving certain problems. In this section, we define these assumptions and problems, explain our notations and introduce the background of the key techniques that will be useful in the sequel.

The symbol || will denote the concatenation of two strings.

The symbol ε will denote the empty string.

The notation ‘ $x \in_R E$ ’ means that *x* is chosen uniformly at random from the set *E*.

The notation ‘ $x \overset{?}{=} y$ ’, used in a protocol, means that the party must check whether *x* is equal to *y*. It is assumed that if the verification fails, the protocol stops.

H will denote a collision-resistant hash function.

2.1 The Representation Problem in a Group of Prime Order

For convenience, we use in this paper, as a group of prime order, the cyclic subgroup G_q of order *q* of Z_p^*, where *q* and *p* are two large primes such that $q / p-1$.

The representation problem in G_q is the following:

Given as inputs a k-tuple $\left(g_1, g_2, \ldots, g_k\right)$ of distinct generators of G_q and $h \in G_q$, the problem is to find a tuple $\left(x_1, x_2, \ldots, x_k\right)$, with $x_i \in Z_q$ for all $1 \leq i \leq k$ such that $h=\prod_{i=1}^{k} g_i^{x_i}$.

$\left(x_1, x_2, \ldots, x_k\right)$ is called a representation of h with respect to $\left(g_1, g_2, \ldots, g_k\right)$.

Remark. For *h* = 1, there is a *trivial* representation namely (0,0,...,0). It is believed to be difficult to solve the representation problem for randomly chosen inputs. In fact, the hardness of finding a representation for random elements is based on the difficulty in computing discrete logarithms in G_q (for more details, we refer interested readers to [2]). A consequence of the difficulty in solving the representation problem is that only one representation of a (non random) element *h*, with respect to a random tuple of generators, can be known (see [2] for more details).

2.2 The Decision-Diffie-Hellman Problem

The Decision-Diffie-Hellman problem is defined as follows: given as inputs g_1, g_2, g_3 and g_4 four elements of G_q, decide whether $\log_{g_4} g_3 = \log_{g_4} g_1 \times \log_{g_4} g_2$. In other words, decide whether $\log_{g_2} g_3 = \log_{g_4} g_1$. For randomly chosen inputs, the Decision-Diffie-Hellman problem is assumed to be difficult to solve.

2.3 Proofs of Knowledge

In this section, we define some well-known cryptographic primitives closely related to the above supposedly hard problems.

In the sequel, we will need a proof of knowledge of a representation and a proof of equality of two discrete logarithms. As proof of knowledge of a representation, we are inspired by a proof due to T. Okamoto [16]. For the proof of equality of two discrete logarithms, we use the proof described in [9].

Definition 1 (*Proof*$_{REP}$) A (message-dependent) proof of knowledge of a representation of h with respect to $(g_1, g_2, ..., g_k)$ is the $k+1$-tuple $(c, r_1, r_2, \ldots, r_k) = Proof_{REP}(M, g_1, g_2, \ldots, g_k, h)$, where M is a message associated to the proof[1], and $c = H(M \| g_1 \| g_2 \|....\| g_k \| h \| g_1^{r_1}\ g_2^{r_2} \cdots g_k^{r_k}\ h^c)$.

The prover who knows the representation $(x_1, x_2, ..., x_k)$ of h with respect to $(g_1, g_2, ..., g_k)$ can construct such a proof.

For this purpose, he chooses k random numbers $(a_1, a_2, \ldots, a_k) \in_R Z_q^{*k}$ and computes $c = H(M \| g_1 \| g_2 \|....\| g_k \| h \| g_1^{a_1}\ g_2^{a_2} \cdots g_k^{a_k})$. Then, he computes $r_i = a_i - c\, x_i \bmod q$ for $1 \le i \le k$. To verify such a proof, the verifier checks whether c is equal to $H(M \| g_1 \| g_2 \|....\| g_k \| h \| g_1^{r_1}\ g_2^{r_2} \cdots g_k^{r_k}\ h^c)$.

Note: According to the definition of [13], Proof$_{REP}$ is not a proof of knowledge. However, it is assumed that this proof does not leak any information about the representation that the prover knows.

Definition 2 (*Proof*$_{LOGEQ}$) A (message-dependent) proof of equality of the discrete logarithm of h with respect to g and of h' with respect to g' is a tuple $(c, r) = Proof_{LOGEQ}(M, g, h, g', h')$ where, as before, M is a (possibly empty) message associated to the proof, and $c = H(M \| g \| h \| g' \| h' \| g^r\ h^c \| g'^r\ h'^c)$.

This proof can be obtained, if and only if the prover knows the discrete logarithms $\log_g h$ and $\log_{g'} h'$ and if these values are equal. To construct the proof, the prover chooses $a \in_R Z_q$ and computes $c = H(M \| g \| h \| g' \| h' \| g^a \| g'^a)$ and $r = a - c\, x \bmod q$ (where x denotes $\log_g h$ and also $\log_{g'} h'$). To verify such a proof, the verifier checks whether c is equal to $H(M \| g \| h \| g' \| h' \| g^r\ h^c \| g'^r\ h'^c)$.

[1] Since the proof involves the message M, it is called *message-dependent*. This message may be the empty string ε.

Note: It is assumed that this proof does not leak any information about *x*.

For some purposes it is more efficient to merge both above defined proofs, which leads to the following definition:

Definition 3 (*Proof*$_{REP+LOGEQ}$) A (message-dependent) proof of knowledge of a representation of h with respect to $(g_1, g_2, ..., g_k)$, which also proves that the exponent of g_1 in this representation is equal to $\log_{g_1'} h'$ is a *k+1*-tuple $(c, r_1, r_2, ..., r_k) = Proof_{REP+LOGEQ}(M,\ g_1,\ g_2, ..., g_k, h, g_1', h')$, where c is equal to $H(M \,||\, g_1 \,||\, g_2 \,||\, ... \,||\, g_k \,||\, h \,||\, g_1' \,||\, h' \,||\, g_1^{r_1}\ g_2^{r_2} \cdots g_k^{r_k}\ h^c \,||\, g_1'^{r_1}\ h'^c)$.

The prover who knows the representation $(x_1, x_2, ..., x_k)$ of h with respect to $(g_1, g_2, ..., g_k)$ and $\log_{g_1'} h' = x_1$ can construct such a proof. For this purpose, he chooses k random numbers $(a_1, a_2, ..., a_k) \in_R Z_q^{*k}$ and computes $c = H(M \,||\, g_1 \,||\, g_2 \,||\, ... \,||\, g_k \,||\, h \,||\, g_1' \,||\, h' \,||\, g_1^{a_1}\ g_2^{a_2} \cdots g_k^{a_k} \,||\, g_1'^{a_1})$. Then he computes $r_i = a_i - c\ x_i \bmod q$ for $1 \le i \le k$. The verifier of this proof checks whether c is equal to $H(M \,||\, g_1 \,||\, g_2 \,||\, ... \,||\, g_k \,||\, h \,||\, g_1' \,||\, h' \,||\, g_1^{r_1}\ g_2^{r_2} \cdots g_k^{r_k}\ h^c \,||\, g_1'^{r_1}\ h'^c)$.

2.4 The Basic Signature Scheme

In our system, the withdrawn coins are signed by the bank using the signature scheme presented in [9]. We now briefly describe this scheme.

The Parameters. Before the scheme can be used, the following parameters must be generated:

1. p and q are two large primes such that $q / p - 1$.
2. g is a generator of G_q.
3. $x \in Z_q^*$ is the signer's secret key, and (p, q, g, h), where $h = g^x$, is the signer's public key.
4. $m \in G_q$ is the message to be signed, and $M \in \{0,1\}^*$ is another (possibly empty) message associated to the signature, for which it is called *message-dependent*.

The Scheme. The message-dependent signature *Sig(m)* on m consists of $z = m^x$ along with a proof that $\log_g h = \log_m z$.

So, we have $Sig(m) = (\ z,\ Proof_{LOGEQ}(M, g, h, m, z)\ ) = (z, c, r)$.

The verifier of such a signature checks whether c is equal to $H(M \,||\, g \,||\, h \,||\, m \,||\, z \,||\, g^r\ h^c \,||\, m^r\ z^c)$.

Note: In the sequel, we will deliberately omit the fixed values *g* and *h* in the computation of *c*. For a discussion of the security of this scheme, we refer interested readers to [9].

The Blind Signature Scheme. The signature scheme described in the previous section can be transformed into a blind signature scheme using Ohta-Okamoto's techniques [17]. To get a blind signature on the message $m \in G_q$, one chooses a random $s \in Z_q^*$ and asks the signer to sign $m_0 = m\, g^s$ (the input of the protocol). Let $z_0 = m_0^x$. The signer then proves that $\log_g h = \log_{m_0} z_0$, as described in figure 1 below.

We will denote this protocol *BlindSig(M, m*$_0$*)*. The following two propositions from [9] show that *BlindSig* is really a blind signature scheme.

Proposition 1 (correctness). If both parties follow the protocol, then *Sig(m)* will be a correct signature on m.

Proposition 2 (blindness). The signer will get no information about m and *Sig(m)*, if the verifier follows the protocol.

The security of our fair electronic cash relies on some assumptions about *BlindSig*. Let us formulate these assumptions.

Assumption 1 (unforgeability). Several executions of *BlindSig* will not help a verifier to create more valid message-signature pairs than expected (i.e., from *l*, sequential or parallel, executions of *BlindSig* with the signer, the verifier cannot create more than *l* valid message-signature pairs). Furthermore, by executing *BlindSig*, the verifier obtains no useful information about the signer's secret key.

Assumption 2 (blinding restrictions). On input m_0 to *BlindSig*, the verifier can obtain a valid blind signature on a message $m \in G_q$ (if and) only if he knows a representation of m with respect to (m_0, g) [1].

It is not hard to prove (using the same technique as in *BlindSig*) that on input $m_0 \in G_q$ to *BlindSig*, the verifier can choose a tuple $(\alpha, \beta) \in_R Z_q$ and obtain a blind signature on $m = g^\alpha\, m_0^\beta$.

Any way of obtaining a valid signature on a message *m*, for which the verifier does not know the representation with respect to (m_0, g), would provide a new method for blinding this kind of signatures.

In our fair cash scheme which uses *BlindSig* as a subprotocol, we will further 'restrict' the blinding manipulations that the user can do. More precisely, we will 'constraint' the user to choose a tuple (α, β) with $\beta = 1$.

[1] The security of Brands'electronic cash system [3] is based on the same assumption.

Signer		Verifier
		choose $s \in_R Z_q^*$
		compute $m_0 = m\, g^s$
	$\xleftarrow{m_0}$	
choose $\omega \in_R Z_q^*$		
compute $z_0 = m_0^x$		
$A_0 = g^\omega$		
$B_0 = m_0^\omega$		
	$\xrightarrow{z_0,\, A_0,\, B_0}$	
		choose $u, v \in_R Z_q^*$
		compute $A = A_0^u\, g^v$
		$B = B_0^u\, m_0^v \big/ A^s$
		$z = z_0 \big/ h^s$
		$c = H(M, m, z, A, B)$
		$c_0 = c/u \bmod q$
	$\xleftarrow{c_0}$	
compute $r_0 = \omega - c_0 x \bmod q$		
	$\xrightarrow{r_0}$	
		verify $A_0 \overset{?}{=} g^{r_0}\, h^{c_0}$
		$B_0 \overset{?}{=} m_0^{r_0}\, z_0^{c_0}$
		compute $r = u\, r_0 + v \bmod q$
		$Sig(m) = (z, c, r)$

Figure 1 : The blind signature protocol *BlindSig(M, m₀)*

Note: From the mere Chaum and Pedersen signature protocol of [9], we obtain the blinding of the *BlindSig* protocol as follows:

From $\begin{cases} A_0 = g^{r_0}\, h^{c_0} \\ B_0 = m_0^{r_0}\, z_0^{c_0} \end{cases}$, we first blind (r_0, c_0) with (u, v) and obtain $\begin{cases} A_0' = A_0^u\, g^v = g^r\, h^c \\ B_0' = B_0^u\, m_0^v = m_0^r\, z_0^c \end{cases}$, where $\begin{cases} r = u\, r_0 + v \\ c = u\, c_0 \end{cases}$. Then we blind (m_0, z_0) with (α, β), which leads to $\begin{cases} A = A_0' = g^r \ h^c \\ B = A_0'^{\alpha} B_0'^{\beta} = m^r \ z^c \end{cases}$, where $\begin{cases} m = g^{\alpha} m_0^{\beta} \\ z = h^{\alpha} z_0^{\beta} \end{cases}$. We finally put $\begin{cases} \alpha = -s \\ \beta = 1 \end{cases}$, which corresponds to the required blinding restriction $m_0 = m\, g^s$. Note that the *BlindSig* protocol does not enforce $\beta = 1$, but this property will be verified when the signature is presented, so the verifier must set $\beta = 1$ in order to get a valid signature. The actual calculation is performed in reverse order, as shown in the *BlindSig* protocol.

3 A Generic Fair Electronic Cash System

3.1 The Setup of the System

The Parameters. For the sake of simplicity, we assume that there is only one coin denomination in the system (extension to multiple denominations is easy).
Two large primes p and q such that $q / p - 1$ are generated. Let G_q be the unique subgroup of Z_p^* of order q.

The Trusted Authority. We assume that there is only one trusted authority T (extension to several trustees is easy).

1. T chooses two secret values x_T and y_T of Z_q^*, and a public value $g_T \in G_q$.
2. T publishes g_T, $h_{CT} = g_T^{x_T^{-1}}$ and $h_{OT} = g_T^{y_T^{-1}}$.

The Bank. Before issuing coins, the bank has to make publicly known some parameters.

1. First, B generates at random her secret key $x \in Z_q^*$.
2. B generates the three generators g, g_1, g_2 of G_q (it is assumed that no representation of either of these elements with respect to the others is known), and publishes the corresponding public values $h = g^x$ (the bank's public key), $h_1 = g_1^x$, and $h_2 = g_2^x$.
3. B publishes $h_T = g_T^x$.
4. B finally determines a collision-free hash function H that maps $\{0,1\}^*$ to Z_{2^k}, where k is an appropriate security parameter.

Opening an Account (performed for each new user *U*). When a user U opens an account at the bank B, he generates at random his secret key $x_u \in Z_q^*$ and computes $Id_U = g_1^{x_u}$ (U's account number).

Then, U must prove to B that he knows the representation of Id_U with respect to g_1. For this purpose, U computes a $Proof_{REP}(\varepsilon, g_1, Id_U)$ (or uses Schnorr's identification scheme [18]) and sends this proof to B. B verifies this proof and if the verification is successful, stores Id_U in the new user's entry of the account database. U and B then independently compute $P_U = h_1^{x_u} = (Id_U)^x$.

3.2 The Withdrawal Protocol

Before the user U and the bank *B* begin the protocol, U must authenticate itself to *B*, so that *B* is sure that *U* is the owner of the account Id_U. This can be done by any (fast) standard authentication protocol.

The withdrawal consists of two phases: the coin tracing phase and the coin withdrawal phase. During the coin tracing phase, the user gives the bank the information that will enable the trusted authority to recognize the withdrawn coin after it has been spent. In the coin withdrawal phase, the user executes *BlindSig* with the bank in order to obtain a blind signature on his coin.

Step 1 - The Coin Tracing Protocol. Roughly, the user generates a 'verifiable' El-Gamal encryption [12], computed with *T's* public key, of the value that will be used to blind the input of *BlindSig*. By verifiable we mean that the user gives a proof to the bank that he has really encrypted this value (without revealing it).

In the sequel, we will put $F = g_T\ g$.

1. *U* chooses $s \in_R Z_q$, computes $G = F^s = g_T^s\ g^s$ and the '*cointrace*' $ct = h_{CT}^s$ [1].

   U generates $Proof(ct) = Proof_{LOGEQ}(r_a, F, G, h_{CT}, ct)$ in order to convince *B* that $\log_F G$ and $\log_{h_{CT}} ct$ are equal. The message r_a is provided for linking the withdrawal to the prior authentication and represents for instance the last user response of the authentication protocol.

   U sends *ct, G,* and *Proof* (*ct*) to *B.*
2. The bank verifies this proof and if the verification holds, stores *ct* in the user's entry of the withdrawal database for possible later anonymity revocation.

[1] g^s will blind the input of *BlindSig,* while g_T^s computationally blinds this blind factor.
The tuple (*ct, G*) can be seen as an El-Gamal encryption, computed with *T*'s public key (h_{CT}, g_T), of the blind factor $Bf = g^s$.

Step 2 - The Withdrawal of a Coin

1. Both the user and the bank prepare the execution of the *Blindsig* protocol by computing independently the blinded coin $Blindcoin = Id_U \times g_2 \times G$, where *B* has got *G* from step 1. Note that $Blindcoin$ = $coin \times g^s$, where $coin = Id_U \times g_2 \times g_T^s$ is the message to be (blindly) signed. As we will see later on (section 3.3), the value g_2 is used to *restrict* the blind manipulations that the user can do (see also the discussion about assumption 2). Note also that *U* need not get z_0 (see figure 1) from *B* in order to compute $z = coin^x = P_U \; h_2 \; h_T^s$.

2. The bank and the user execute a *message-dependent BlindSig* protocol *BlindSig(M, Blindcoin),* where the message *M* = *ot* || *D* || *E* inserted by *U* (and disclosed during the payment protocol) is intended to both assure *owner-tracing* (thanks to the '*ownertrace*' $ot = h_{OT}^s$) and prevent future *double-spending* of the coin (thanks to $D = g_1^a \; g_T^b$ and $E = h_{OT}^b$ where $(a,b) \in_R Z_q^{*2}$ are chosen by *U*).

So at the end of *BlindSig*, *U* obtains $Sig(coin) = (\, z,\ Proof_{LOGEQ}(M,\ g,\ h,\ coin,\ z)\,)$, and the bank debits the real money counterpart of the withdrawn coin from *U's* account.

3.3 The Payment Protocol

We assume that the shop *S* is known under Id_S (its account number for example), and define '*t*' to be the payment (date and) time. During the payment protocol, the user *U* sends the coin signed by the bank to the shop, along with a proof of knowledge of the representation of $C = coin/g_2 = g_1^{x_u} \; g_T^s$ with respect to (g_1, g_T). This proof also provides the shop with the information (*ot*) that will enable the trusted authority to possibly trace him. The message *msg* = (Id_S || *t*) is inserted in this proof in order to detect theft (by another shop) and replay (that is, multiple spending by the user, as explained in 3.5) of the coin.

1. *U* uses the commitments *D* and *E* to generate :
   $Proof\,(ot) = Proof_{REP+LOGEQ}(msg,\ g_T,\ g_1,\ C,\ h_{OT},\ ot)$.
   More precisely, *Proof* (*ot*) = (c, r_1, r_2), where:
   . $c = H(msg \,||\, g_T \,||\, g_1 \,||\, C \,||\, h_{OT} \,||\, ot \,||\, D \,||\, E)$
   . $r_1 = b - c\ s \mod q$ and $r_2 = a - c\ x_u \mod q$, with (*a, b*) from the withdrawal (see section 3.2 Step 2).
   U sends *M, coin, Sig* (*coin*) and *Proof* (*ot*) to the shop.

2. *S* verifies the signature and the proof and, if the verification holds, accepts the payment.

Roughly speaking, *U* generates a 'verifiable' El-Gamal encryption of Id_U, computed with *T*'s public key (h_{OT}, g_T).

3.4 The Deposit Protocol

To be credited the value of this coin, the shop sends the transcript of the execution of the payment protocol to the bank, which verifies, exactly as the shop did, that the coin ($coin$) bears the bank's signature and that the other responses are correct.

3.5 The Tracing Mechanisms

Double-Spenders Tracing (without having recourse to T). If the user U spends the same coin twice, either in two different shops S and S' (in which case $Id_S \neq Id_{S'}$), or in the same shop but at two different times *t* and *t'*, the messages *msg* and *msg'* associated to these payments will always be different, and so will be the values *c* and *c'* with high probability. After the deposit of both payment transcripts, the bank will be able to detect the double spending of the coin (thanks to *coin*) *and* to retrieve the double spender's identity, that is x_u, and therefore U's account number ($Id_U = g_1^{x_u}$). Indeed, from ($c = H(msg \| g_T \| g_1 \| C \| h_{OT} \| ot \| D \| E)$, $r_2 = a - c\,x_u$ mod q) received during the one deposit and ($c' = H(msg' \| g_T \| g_1 \| C \| h_{OT} \| ot \| D \| E)$, $r_2' = a - c'\,x_u$ mod q) received during the other, the bank can compute $x_u = \frac{r_2' - r_2}{c - c'}$ mod q.

Coin Tracing. The coin tracing mechanism allows the trusted authority to compute the information needed to recognize the money after it has been spent.

Given the transcript of the execution of the withdrawal protocol (in fact *ct*), the trusted authority can compute $g_T^s = (ct)^{x_T}$ and then $coin = Id_U\ g_2\ g_T^s$ (so, the coin is traceable).

Owner Tracing. The owner tracing mechanism allows the trusted authority to find the identity of the person (in fact the account number) who has withdrawn a specific coin.

Given the execution transcript of the payment protocol (in fact $coin$ and *ot*), the trusted authority can retrieve U's account number: $Id_U = (coin/g_2)/(ot)^{y_T}$.

3.6 The Security of the Scheme

Let us analyze, informally speaking, the security of our scheme.

Correctness. If *U* and *B* behave correctly, then *U* will obtain a valid signature of *coin* (correctness of the protocol *BlindSig*, see Proposition 1). Moreover, if *U* behaves correctly, then he knows *s* and x_u and will be able to generate the two valid proofs $Proof(ct)$ and $Proof(ot)$ (see Definition 2 and 3). So, *S* will accept the payment.

Security for B (unforgeability). We refer to the first assumption about *BlindSig*, which says that it is hard to forge a signature of our basic signature scheme even after several sequential or parallel executions of *BlindSig.* Moreover, in the withdrawal protocol the bank executes only *BlindSig*, so the withdrawal protocol does not leak more useful information about the bank's secret key than *BlindSig*.

So, it seems difficult to get more distinct coins than the ones that have been withdrawn.

Anonymity. Let us now explain why our system provides anonymity to the users[1].

From proposition 2 we know that *BlindSig* is a perfect blind signature scheme, that is, the signer's *view*[2] of *BlindSig* and the verifier's output of *BlindSig* (the message-signature pair) are unlinkable.

So, in our cash system only the extra parts of *BlindSig* (namely, $G = F^s$, $ct = h_{CT}^s$, and $ot = h_{OT}^s$) could help the bank to link its view of the withdrawal protocol to the coin obtained (and spent) by the user. As the signer *B* knows neither $\log_{g_T} h_{OT}$ nor $\log_{g_T} h_{CT}$, these extra parts are computationally unlinkable (see the Decision-Diffie-Hellman problem section 2.2).

Moreover, $Proof(ct)$ and $Proof(ot)$ leak no information that seems useful for establishing a link between a withdrawal and a payment.

Security for T. We explain why it is infeasible for a user to get a valid coin, which, if spent, would not allow *T* to trace him (i.e. it is impossible to 'bypass' the tracing mechanisms):

1. The proof $Proof(ct)$ guarantees that the input of *BlindSig* in the withdrawal protocol (*Blindcoin*) is of the form: $g_1^{x_u}\ g_2\ g_T^s\ g^s$ with $s \in Z_q$.
2. According to Assumption 2, on input *Blindcoin* to *BlindSig*, *Sig(coin)* is a valid (blind) signature on *coin*, only if the user knows a representation $(\alpha, \beta) \in Z_q$ of

[1] which can only be computational in fair cash system (as proved in [14]).
[2] A view consists of the complete list of data seen during the execution of a protocol.

coin with respect to $(g, Blindcoin)$. So *coin* is necessarily of the form $coin = g^{\alpha} g_1^{\beta x_u} g_2^{\beta} g_T^{\beta s} g^{\beta s}$.

3. The proof $Proof(ot)$ guarantees that the user knows a representation $(\gamma, \delta) \in Z_q$ of $coin/g_2$ with respect to (g_1, g_T). So, *coin* is also of the form: $coin = g_1^{\gamma} g_2 g_T^{\delta}$.

Since the user knows at most only one representation of *coin* with respect to (g_1, g_2, g, g_T) (see the remark in section 2.1), this implies that $\begin{cases} \alpha = -s \\ \beta = 1 \end{cases}$ (as set in the end of section 2.4) and $\begin{cases} \gamma = x_u \mod q \\ \delta = s \mod q \end{cases}$.

As $Proof(ct)$ guarantees that $s = \log_{h_{CT}} ct$ and $Proof(ot)$ guarantees that $\delta = \log_{h_{OT}} ot$, and since these values are necessarily equal, this implies that the coin tracing and the owner tracing are possible.

Efficiency. Let us now briefly compare the efficiency of our scheme with that of [11]. Roughly, in the payment protocol of [11], the user and the shop perform each 11 more exponentiations than in our system. In the deposit protocol of [11], the bank performs on the order of 9 more exponentiations than in our system. The withdrawal protocol of [11] is about as efficient as ours.

4 Extension to Wallet with Observer

In our basic fair off-line cash system, the multiple spending of coins can only be detected but not prevented. In [9], Chaum and Pedersen introduced the concept of ‘electronic wallet with observer’ as a way to offer prior restraint of multiple spending in off-line anonymous electronic cash systems, without compromising the untraceability of payments.

An observer is a tamper-resistant device, issued by the bank (so, not necessarily trusted by the user), and inserted into the payment device of the user. Transactions can only be executed with the cooperation of the observer, which will only cooperate in spending each coin once. The protocols involving an observer are constructed in such a way that the observer cannot ‘leak’ (subliminal) information that would compromise the privacy of the transaction made by the user. Moreover, the transactions remain untraceable even if it is possible to analyze the information collected by the observer when it was involved (except if the observer has an internal clock).

It is possible in observer-based systems, to use a double-spender tracing mechanism (like in our generic cash system), which will serve in this case as a second level of protection useful in case the tamper-resistance of an observer has been broken (for a thorough discussion about wallet with observers, see [9, 10]).

In this section, we describe how to extend our generic fair cash system to the setting of a wallet with observers. In this extension, a double-spender tracing mechanism is incorporated and the owner tracing provides a link to the account of the user.

4.1 The Setup of the System

The setup of the system is the same as in the generic fair cash system.

Opening an Account. *U* generates at random his secret key $x_u \in Z_q^*$, computes $Id_U = g_1^{x_u}$ and transmits Id_U to *B*.

Then, U proves (using, for example, Schnorr's identification scheme) to B that he knows the representation of Id_U with respect to g_1. B verifies this proof and if the verification is successful, stores Id_U in the new user's entry of the account database.

Then the bank *B* supplies *U* with an observer *O*. *O* holds in its memory a secret key $x_o \in_R Z_q^*$ which is unknown to *U* (each observer has its own secret key). Let us denote $Id_O = g_1^{x_o}$ and $P_O = Id_O^x$. *B* computes $Id'_U = Id_O\ Id_U$ and $P'_U = Id'^x_U$, and transmits these values to *U*. Id'_U will designate *U*'s account number (it should be noted that *U* does not know $\log_{g_1} Id'_U = x_o + x_u$). We will put $x'_u = \log_{g_1} Id'_U$.

The Withdrawal Protocol. The withdrawal protocol is very similar to the withdrawal protocol of the generic fair cash system, except that (x_u, Id_U) is replaced by (x'_u, Id'_U) and that the user and the observer 'jointly' compute the commitment *D* (see section 3.2.2). More precisely, *D* is computed as follows:

1. *O* chooses $\omega \in_R Z_q^*$ and computes the commitment $A = g_1^{\omega}$. *O* sends *A* to *U*. *O* then stores *A* in its list of active commitments (i.e. not yet used in a payment).

2. *U* chooses $(u,v,b) \in_R Z_q^{*3}$ and computes $D = A^u\ g_1^v\ g_T^b$. As in section 3.2, we have $D = g_1^a g_T^b$, where here $a = u\ \omega + v$.

The Payment Protocol. The payment protocol is very similar to the payment protocol of the generic fair cash system, except that the user and the observer jointly generate the proof $Proof(ot)$ (see section 3.3). Indeed, in order to prevent multiple spending, the participation of *O* is somehow needed.

More precisely, $Proof\,(ot)$ is generated as follows (we use the same notations as in section 3.3):

1. U computes $c = H(msg \,||\, g_T \,||\, g_1 \,||\, C \,||\, h_{OT} \,||\, ot \,||\, D \,||\, E)$ and $c_0 = c/u \mod q$. Then U sends c_0 to O and asks O to prove his knowledge of $\log_{g_1} Id_O$ [1] using A.
2. O verifies that A is on its list of active commitments and if so computes $r_0 = \omega - c_0\, x_o \mod q$. O then sends r_0 to U and erases A from its list of active commitments.
3. U computes $r_1 = b - c\, s \mod q$ and $r_2 = u\, r_0 + v - c\, x_u \mod q$ (also equal to $a - cx'_u$) and sends $Proof\,(ot)$ to the shop, where $Proof\,(ot) = (c, r_1, r_2) = Proof_{REP+LOGEQ}(msg, g_T, g_1, C, h_{OT}, ot)$.
4. The shop verifies the signature and the proof (as in the generic fair cash system) and if the verification holds, accepts the payment.

The deposit protocol is exactly the same as in the generic fair cash system.

The tracing mechanisms follow immediately from those of the generic fair cash system. It should be noted that contrarily to the scheme of [6], the owner tracing in our scheme yields the account number of the user Id'_U and not only a link to the withdrawal database.

Security of this Extension. If we view O and U as one party, then this extension does not differ from the generic fair cash system. Consequently, the discussion concerning the security for B, the security for T and the correctness of the generic fair cash system, also applies to this extension.

Under our assumptions, it can be shown that the withdrawn coin (*coin*) is necessarily of the form: $coin = g_1^{x'_u}\; g_2\; g_T^{s}$ with $s \in Z_q$.

In order to be able to pay the shop with this coin, the user must know a representation of $coin/g_2$ with respect to (g_1, g_T) . Since the user does not know by himself x'_u (recall that $x'_u = x_o + x_u \mod q$), he cannot spend this coin without the cooperation of O [2].

[1] For this purpose O and U will perform the Schnorr's identification protocol [18]. In fact, the protocol that they will execute, referred to as *BlindSchnorr* in the sequel, is a blind issuing protocol for Schnorr signatures [16].

[2] We implicitly assume that the tamper-resistance of O has not been broken. In this extension, O executes only *BlindSchnorr*. So, we also assume that it is hard to forge a Schnorr-like signature (even after several sequential or parallel executions of *BlindSchnorr*) and that U obtains no useful information about O's secret key by executing *BlindSchnorr* with O.

As the user blinds all the values sent by *O* (recall that the observer proves that he knows $\log_{g_1} Id_O$ in a blind manner), the observer cannot compromise the privacy of the transaction made by *U* (even if the bank can obtain the observer, for maintenance purposes for example, and analyze its contents).

Efficiency of this Extension. Let us now briefly compare the efficiency of our extension with that of [6]. Roughly, in [6] the observer performs 9 times more computations than in our system, while the global workload of the other entities is roughly the same in the two systems.

Note: we can reduce (by a factor 2) the computations of *U* and *S* in the payment protocol, if we only require a link to the withdrawal database for the owner tracing.

5 Extension to Checks

A drawback of off-line coins based-systems (fair or not) is that they are not practical (in terms of computation, communication and storage requirements) when amounts to be paid require several coins. Moreover, for privacy reasons, anonymous off-line electronic coins schemes do not provide means to give change in a payment transaction. This implies that a user may have enough money to perform a transaction but be unable to pay because he has not the correct change.

A more convenient means of payment is the electronic check. An electronic check can be used for any amount up to a maximum value and then returned to the bank for a refund of the unspent part (so, an extra protocol, the *refund protocol* is needed). In this section, we describe how to extend our generic fair cash system to a fair electronic check system.

5.1 The Setup

For the sake of simplicity, in our description the owner tracing provides only a link to the transcript of the withdrawal of the check (see section 5.5). We will make the following modifications:

1. The trustee *T*, unlike 3.1, chooses $x_T, y_T \in_R Z_q$, and publishes $G_{CT} = g^{x_T^{-1}}$ and $G_{OT} = G_{CT}^{y_T^{-1}}$.
2. The bank *B*, in addition to 3.1, makes publicly known $(d_1, d_2, ..., d_k)$ a *k*-tuple of randomly chosen generators of G_q, and a defined amount of money associated to each of these generators; in our description, we assume that the bank assigns a value of 2^{i-1}\$ to d_i. *B* also publishes the *k* values $D_i = d_i^x$, for $1 \le i \le k$.

5.2 The Withdrawal of a Fair Check

1. To withdraw a (fair) check that can be spent up to $2^k - 1$ \$, *U* generates *k* random values $(a_1, a_2, ..., a_k) \in_R Z_q^{*k}$, called the *payment terms*, and $s \in_R Z_q^*$, and computes $G = F^s \prod_{i=1}^{k} d_i^{a_i}$.

   U generates $Proof(ct) = Proof_{REP+LOGEQ}(r_a, F, d_1, d_2, ..., d_k, G, G_{CT}, ct)$, where $ct = G_{CT}^s$, and r_a was defined in 3.2, and sends *G, ct,* and *Proof* (*ct*) to *B.*

2. The bank verifies this proof and, if the verification holds, stores (*G*, *ct*) in the user's entry of the cheque database (otherwise stops the protocol).

3. Both the user and the bank prepare the execution of the *BlindSig* protocol by computing independently $Blindcheck = Id_U \times g_2 \times G$[1]. Note that $Blindcheck = check \times g^s$, where $check = Id_U\ g_2\ g_T^s \prod_{i=1}^{k} d_i^{a_i}$ is the message to be (blindly) signed. Note also that *U* need not get z_0 (see figure 1) from *B* to compute $z = check^x = P_U\ h_2\ h_T^s \prod_{i=1}^{k} D_i^{a_i}$

4. *U* and *B* then execute the *BlindSig(M, Blindcheck)* in order for *U* to get the (blind) signature $Sig(check) = (z, Proof_{LOGEQ}(M, g, h, check, z))$, where $M = D \,||\, E \,||\, ot$ is intended to detect multiple spending of the check. To compute the commitments *D* and *E*, and the *ownertrace ot*, the user *U* chooses $(b_1, b_2, ..., b_k, a, b) \in_R Z_q^{*(k+2)}$, and puts $D = d_1^{b_1}\ d_2^{b_2} \cdots d_k^{b_k}\ g_1^a\ g_T^b$, $E = G_{OT}^b$, and $ot = G_{OT}^s$. Finally, the bank debits $2^k - 1$ \$ from *U's* account.

5.2 The Payment Protocol

Let *K* be the subset $\{1, \cdots, k\}$ of *N*. Suppose the user wishes to spend in the shop *S* a certain amount of money $\sum_{j \in J} 2^{j-1}$, where $J \subset K$.

1. *U* reveals to *S* the values *check, Sig(check),* $(a_j)_{j \in J}$ and $(b_j)_{j \in J}$.

2. *U* proves to *S*, using the commitments $D \big/ \prod_{j \in J} d_j^{b_j}$ and *E*, that he knows a representation of $C = (check/g_2) \big/ \prod_{j \in J} d_j^{a_j}$ with respect to $(g_T, g_1, (d_i)_{i \in K \setminus J})$, and

[1] In fact, the factor Id_U is only necessary if a direct owner tracing is required (see section 5.5 for more details).

that the exponent of g_T in this representation is equal to $\log_{G_{OT}} ot$. Doing so, *U* proves that he is the owner of *check* and gives the shop some useful information that will enable the trusted authority to trace him.

3. *S* verifies the signature *Sig(check)* and the proof of knowledge, and if the verifications hold, accepts the payment.

5.4 The Deposit and Refund Protocols

1. *S* sends the transcript of the execution of the payment protocol to the bank *B*, who (after verifying its correctness) stores the a_j for $j \in J$ on a list, that we will call the *refund list*.

When *U* wishes to get a refund of the unspent part $\sum_{i \in K \setminus J} 2^{i-1}$ of the check,

2. *U* sends the unspent payment terms $\left(a_i\right)_{i \in K \setminus J}$ to the bank, along with *G* and a proof of knowledge of a representation of $G \Big/ \prod_{i \in K \setminus J} d_i^{a_i}$ with respect to $(F, \left(d_j\right)_{j \in J})$.
3. The bank *B* verifies that the user's entry in the cheque database holds *G*, and also that the a_i for $i \in K \setminus J$ are not already on the refund list. Then *B* verifies the proof, and if the verification holds, *B* refunds the user the corresponding amount of money, erases *G* from the cheque database, and stores the a_i for $i \in K \setminus J$ on the refund list.

The refund list maintained by *B* prevents the refunding of spent payment terms, and also the spending of (already) refunded payment terms. After the deposit of a check, the user is excluded not only if he spent the check twice, but also if he had already got a refund of some spent payment terms.

5.5 Tracing Mechanisms, Security, Efficiency

The double-spender tracing follows immediately from that of the generic fair cash system. The check tracing mechanism consists for *T* in computing $check = Blindcheck \big/ ct^{x_T}$, and, as said in section 5.1, to simplify our description the owner tracing mechanism provides only a link to the withdrawal database: that is, with *ot*, *T* can compute $ct = ot^{y_T}$, which was stored during the withdrawal of this check in the user's entry of the cheque database, and therefore *T* and *B* can retrieve *U's* identity.

The proof of correctness of this extension as well as the other security requirements follow immediately from those of the generic fair cash system.

The fair check system described here is more efficient than the *basic* (not fair) check system of Brands [2]. Indeed, our system provides checks which are more efficiently computed (twice as fast as checks in [2]) and which also require less memory for their storage (half as much).

6 Conclusion and Open Problems

We have proposed an efficient discrete logarithm based fair payment system that is more efficient than that of [11]. Our system supports extensions to wallets with observers and electronic checks which are more efficient than previous ones [2, 6].

The security of the proposed schemes relies on (trustworthy) assumptions about the blind signature scheme that we use. It would be interesting to prove these assumptions.

It is an open problem to find an efficient (avoiding in particular the cut and choose technique) fair payment system based on the factorization problem.

Acknowledgments

We would like to thank Marc Girault and the anonymous referees for their useful comments.

References

1. B. den Boer, D. Chaum, E. van Heyst, S. Mjolsnes and A. Steenbeek, Efficient Off-Line Electronic Checks, *Proceedings of* EUROCRYPT’89, Lecture Notes in Computer Science, Vol 434, Springer-Verlag, pp. 294-301.
2. S. Brands, An Efficient Off-Line Electronic Cash System based on the Representation Problem, Technical Report CS-R9323, CWI, April 1993.
3. S. Brands, Untraceable Off-Line Cash in Wallets with Observers, *Proceedings of* CRYPTO'93, Lecture Notes in Computer Science, Vol 773, Springer-Verlag, pp. 302-318.
4. E. Brickell, P. Gemmel and D. Kravitz, Trustee-Based Tracing Extensions to Anonymous Cash and the Making of Anonymous Change, *Proceedings of* the 6th Annual Symposium on Discrete Algorithm, pp. 457-466, Jan 1995.
5. J. Camenisch, U. Maurer and M. Stadler, Digital Payment Systems with Passive Anonymity-Revoking Trustees, *Proceedings of* ESORICS’96, Lecture Notes in Computer Science, Vol 1146, Springer-Verlag, pp. 33-43.
6. J. Camenisch, U. Maurer and M. Stadler, Digital Payment Systems with Passive Anonymity-Revoking Trustees, Journal of Computer Security, volume 5, number 1, IOS Press, 1997.
7. D. Chaum, Blind Signatures for Untraceable Payments, *Proceedings of* CRYPTO’82, Plenum Press, 1983, pp. 199-203.

8. D. Chaum, A. Fiat and M. Naor, Untraceable Electronic Cash, *Proceedings of* CRYPTO'88, Lecture Notes in Computer Science, Vol 403, Springer-Verlag, pp. 319-327.
9. D. Chaum and T. Pedersen, Wallet Databases with Observers, *Proceedings of* CRYPTO'92, Lecture Notes in Computer Science, Vol 740, Springer-Verlag, pp. 89-105.
10. R. Cramer and T. Pedersen, Improved Privacy in Wallets with Observers, *Proceedings of* EUROCRYPT'93, Lecture Notes in Computer Science, Vol 765, Springer-Verlag, pp. 329-343.
11. G. Davida, Y. Frankel, Y. Tsiounis and M. Yung, Anonymity Control in E-Cash Systems, Financial Cryptography’97, Anguilla, British West Indies, February 24-27.
12. T. ElGamal, A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, *Proceedings of* CRYPTO’84, Lecture Notes in Computer Science, Vol 196, Springer-Verlag, pp. 10-18.
13. U. Feige, A. Fiat and A. Shamir, Zero Knowledge Proofs of Identity, Journal of Cryptology, 1 (2), pp. 77-94, 1988.
14. Y. Frankel, Y. Tsiounis and M. Yung, Indirect Discourse Proofs: Achieving Fair Off-Line Electronic Cash, *Proceedings of* ASIACRYPT’96, Lecture Notes in Computer Science, Vol 1163, Springer-Verlag, pp. 286-300.
15. R. Hirschfeld, Making Electronic Refunds Safer, *Proceedings of* CRYPTO'92, Lecture Notes in Computer Science, Vol 740, Springer-Verlag, pp. 106-112.
16. T. Okamoto, Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes, *Proceedings of* CRYPTO'92, Lecture Notes in Computer Science, Vol 740, Springer-Verlag, pp. 31-53.
17. T. Okamoto and K. Ohta, Divertible Zero-Knowledge Interactive Proofs and Commutative Random Self-Reducibility, *Proceedings of* EUROCRYPT'89, Lecture Notes in Computer Science, Vol 434, Springer-Verlag, pp. 481-496.
18. C.P. Schnorr, Efficient Signature Generation by Smart Cards, Journal of Cryptology, 4(3), pp. 161-174, 1991.
19. M. Stadler, J.M. Piveteau and J. Camenisch, Fair Blind Signatures, *Proceedings of* EUROCRYPT'95, Lecture Notes in Computer Science, Vol 921, Springer-Verlag, pp. 209-219.
20. J. Traoré, Making Unfair a ‘Fair’ Blind Signature Scheme, *Proceedings of* ICICS’97, Lecture Notes in Computer Science, Vol 1334, Springer-Verlag, pp. 386-397.
21. S. von Solms and D. Naccache, On Blind Signatures and Perfect Crimes, Computer & Security, 11, 1992, pp. 581-583.